Veritas™ Volume Manager 5.0 Administrator's Guide

HP-UX 11i v3

First Edition



Manufacturing Part Number: 5992-3942

May 2008

Legal Notices

© Copyright 2008 Hewlett-Packard Development Company, L.P.

Publication Date: 2008

Confidential computer software. Valid license from HP required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

The information contained herein is subject to change without notice. The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

Oracle ® is a registered trademark of Oracle Corporation.

UNIX® is a registered trademark in the United States and other countries, licensed exclusively through The Open Group.

Veritas is a registered trademark of Symantec Corporation.

Contents

Chapter 1 Understanding Veritas Volume Manager

VxVM and the operating system 19
How data is stored 19
How VxVM handles storage management 20
Physical objects—physical disks 20
Virtual objects 26
Volume layouts in VxVM 34
Implementation of non-layered volumes 34
Implementation of layered volumes 34
Layout methods 35
Concatenation and spanning 35
Striping (RAID-0) 38
Mirroring (RAID-1) 42
Striping plus mirroring (mirrored-stripe or RAID-0+1) 42
Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10) 43
RAID-5 (striping with parity) 45
Layered volumes 51
Online relayout 54
How online relayout works 54
Limitations of online relayout 57
Transformation characteristics 58
Transformations and volume length 58
Volume resynchronization 59
Dirty flags 59
Resynchronization process 59
Dirty region logging 60
Dirty region logs 60
Log subdisks and plexes 61
Sequential DRL 61
SmartSync recovery accelerator 62
Volume snapshots 63
Comparison of snapshot features 65
FastResync 66
FastResync enhancements 67
Non-persistent FastResync 67
Persistent FastResync 68

DCO volume versioning 68
FastResync limitations 74
Hot-relocation 75
Volume sets 75

Chapter 2 Administering disks

Disk devices 77
Disk device naming in VxVM 78
Private and public disk regions 80
Discovering and configuring newly added disk devices 82
Partial device discovery 82
Discovering disks and dynamically adding disk arrays 83
Third-party driver coexistence 84
Administering the Device Discovery Layer 85
Placing disks under VxVM control 90
Changing the disk-naming scheme 91
Regenerating persistent device names 93
Changing device naming for TPD-controlled enclosures 94
Discovering the association between enclosure and OS based disk names 94
Issues regarding persistent simple or nopriv disks with enclosure-based naming 94
Installing and formatting disks 96
Displaying and changing default disk layout attributes 97
Adding a disk to VxVM 97
Reinitializing a disk 101
Using vxdiskadd to place a disk under control of VxVM 101
Rootability 102
VxVM root disk volume restrictions 103
Root disk mirrors 103
Booting root volumes 104
Setting up a VxVM root disk and mirror 104
Creating an LVM root disk from a VxVM root disk 106
Adding swap volumes to a VxVM rootable system 107
Adding persistent dump volumes to a VxVM rootable system 107
Removing a persistent dump volume 108
Dynamic LUN expansion 108
Removing disks 110
Removing a disk with subdisks 111
Removing a disk with no subdisks 112
Removing a disk from VxVM control 112
Removing and replacing disks 112
Replacing a failed or removed disk 115
Enabling a disk 117

Taking a disk offline 118
Renaming a disk 119
Reserving disks 119
Displaying disk information 120
Displaying disk information with vxdiskadm 121
Controlling Powerfail Timeout 122
Setting the PFTO values 122
Displaying the PFTO values 122
Enabling or disabling PFTO 123

Chapter 3 Administering dynamic multipathing (DMP)

How DMP works 125
How DMP monitors I/O on paths 128
Load balancing 129
DMP coexistence with HP-UX native multipathing 130
DMP in a clustered environment 132
Disabling and enabling multipathing for specific devices 133
Disabling multipathing and making devices invisible to VxVM 133
Enabling multipathing and making devices visible to VxVM 134
Enabling and disabling I/O for controllers and storage processors 136
Displaying DMP database information 137
Displaying the paths to a disk 137
Administering DMP using vxdmpadm 139
Retrieving information about a DMP node 139
Displaying the members of a LUN group 140
Displaying paths controlled by a DMP node, controller or array port 140
Displaying information about controllers 141
Displaying information about enclosures 142
Displaying information about array ports 142
Displaying information about TPD-controlled devices 143
Gathering and displaying I/O statistics 144
Setting the attributes of the paths to an enclosure 146
Displaying the I/O policy 147
Specifying the I/O policy 147
Disabling I/O for paths, controllers or array ports 153
Enabling I/O for paths, controllers or array ports 154
Upgrading disk controller firmware 154
Renaming an enclosure 155
Configuring the response to I/O failures 156
Configuring the I/O throttling mechanism 157
Displaying recoveryoption values 159
Configuring DMP path restoration policies 160
Stopping the DMP path restoration thread 161

Displaying the status of the DMP path restoration thread 161
Displaying information about the DMP error-handling thread 162
Configuring array policy modules 162

Chapter 4 Creating and administering disk groups

Specifying a disk group to commands 167
System-wide reserved disk groups 167
Rules for determining the default disk group 168
Displaying disk group information 169
Displaying free space in a disk group 170
Creating a disk group 170
Adding a disk to a disk group 171
Removing a disk from a disk group 172
Deporting a disk group 173
Importing a disk group 174
Handling disks with duplicated identifiers 175
Writing a new UDID to a disk 176
Importing a disk group containing cloned disks 176
Sample cases of operations on cloned disks 178
Renaming a disk group 183
Moving disks between disk groups 184
Moving disk groups between systems 185
Handling errors when importing disks 186
Reserving minor numbers for disk groups 187
Compatibility of disk groups between platforms 189
Handling conflicting configuration copies 190
Example of a serial split brain condition in a cluster 190
Correcting conflicting configuration information 194
Reorganizing the contents of disk groups 195
Limitations of disk group split and join 199
Listing objects potentially affected by a move 200
Moving objects between disk groups 203
Splitting disk groups 205
Joining disk groups 206
Disabling a disk group 207
Destroying a disk group 208
Recovering a destroyed disk group 208
Upgrading a disk group 208
Managing the configuration daemon in VxVM 212
Backing up and restoring disk group configuration data 213
Using vxnotify to monitor configuration changes 213

Chapter 5 Creating and administering subdisks

Creating subdisks 215
Displaying subdisk information 216
Moving subdisks 217
Splitting subdisks 217
Joining subdisks 218
Associating subdisks with plexes 218
Associating log subdisks 220
Dissociating subdisks from plexes 221
Removing subdisks 221
Changing subdisk attributes 221

Chapter 6 Creating and administering plexes

Creating plexes 223
Creating a striped plex 224
Displaying plex information 224
Plex states 224
Plex condition flags 228
Plex kernel states 229
Attaching and associating plexes 229
Taking plexes offline 230
Detaching plexes 231
Reattaching plexes 231
Moving plexes 232
Copying volumes to plexes 233
Dissociating and removing plexes 233
Changing plex attributes 234

Chapter 7 Creating volumes

Types of volume layouts 236
Supported volume logs and maps 237
Creating a volume 238
Advanced approach 238
Assisted approach 239
Using vxassist 239
Setting default values for vxassist 241
Discovering the maximum size of a volume 242
Disk group alignment constraints on volumes 242
Creating a volume on any disk 243
Creating a volume on specific disks 244
Specifying ordered allocation of storage to volumes 245
Creating a mirrored volume 249
Creating a mirrored-concatenated volume 249

Creating a concatenated-mirror volume 249
Creating a volume with a version 0 DCO volume 250
Creating a volume with a version 20 DCO volume 252
Creating a volume with dirty region logging enabled 252
Creating a striped volume 253
Creating a mirrored-stripe volume 254
Creating a striped-mirror volume 254
Mirroring across targets, controllers or enclosures 255
Creating a RAID-5 volume 256
Creating tagged volumes 257
Creating a volume using vxmake 258
Creating a volume using a vxmake description file 259
Initializing and starting a volume 260
Initializing and starting a volume created using vxmake 261
Accessing a volume 262

Chapter 8 Administering volumes

Displaying volume information 264
Volume states 265
Volume kernel states 266
Monitoring and controlling tasks 267
Specifying task tags 267
Managing tasks with vxtask 268
Stopping a volume 270
Putting a volume in maintenance mode 270
Starting a volume 271
Adding a mirror to a volume 271
Mirroring all volumes 272
Mirroring volumes on a VM disk 272
Removing a mirror 273
Adding logs and maps to volumes 274
Preparing a volume for DRL and instant snapshots 275
Specifying storage for version 20 DCO plexes 276
Using a DCO and DCO volume with a RAID-5 volume 277
Determining the DCO version number 277
Determining if DRL is enabled on a volume 278
Determining if DRL logging is active on a volume 278
Disabling and re-enabling DRL 278
Removing support for DRL and instant snapshots from a volume ... 279
Upgrading existing volumes to use version 20 DCOs 279
Adding traditional DRL logging to a mirrored volume 281
Removing a traditional DRL log 282
Adding a RAID-5 log 283

Adding a RAID-5 log using vxplex 283
Removing a RAID-5 log 284
Resizing a volume 284
Resizing volumes using vxresize 285
Resizing volumes using vxassist 286
Resizing volumes using vxvol 287
Setting tags on volumes 288
Changing the read policy for mirrored volumes 289
Removing a volume 290
Moving volumes from a VM disk 290
Enabling FastResync on a volume 292
Checking whether FastResync is enabled on a volume 293
Disabling FastResync 293
Performing online relayout 294
Permitted relayout transformations 295
Specifying a non-default layout 298
Specifying a plex for relayout 298
Tagging a relayout operation 298
Viewing the status of a relayout 299
Controlling the progress of a relayout 299
Converting between layered and non-layered volumes 300

Chapter 9 Administering volume snapshots

Traditional third-mirror break-off snapshots 305
Full-sized instant snapshots 307
Space-optimized instant snapshots 309
Emulation of third-mirror break-off snapshots 310
Linked break-off snapshot volumes 311
Cascaded snapshots 312
Creating a snapshot of a snapshot 313
Creating multiple snapshots 317
Restoring the original volume from a snapshot 317
Creating instant snapshots 319
Preparing to create instant and break-off snapshots 321
Creating and managing space-optimized instant snapshots 324
Creating and managing full-sized instant snapshots 327
Creating and managing third-mirror break-off snapshots 329
Creating and managing linked break-off snapshot volumes 331
Creating multiple instant snapshots 333
Creating instant snapshots of volume sets 334
Adding snapshot mirrors to a volume 336
Removing a snapshot mirror 336
Removing a linked break-off snapshot volume 337

Adding a snapshot to a cascaded snapshot hierarchy 337
Refreshing an instant snapshot 337
Reattaching an instant snapshot 338
Reattaching a linked break-off snapshot volume 339
Restoring a volume from an instant snapshot 340
Dissociating an instant snapshot 340
Removing an instant snapshot 341
Splitting an instant snapshot hierarchy 341
Displaying instant snapshot information 342
Controlling instant snapshot synchronization 344
Listing the snapshots created on a cache 345
Tuning the autogrow attributes of a cache 346
Growing and shrinking a cache 347
Removing a cache 347
Creating traditional third-mirror break-off snapshots 348
Converting a plex into a snapshot plex 351
Creating multiple snapshots 352
Reattaching a snapshot volume 352
Adding plexes to a snapshot volume 353
Dissociating a snapshot volume 354
Displaying snapshot information 355
Adding a version 0 DCO and DCO volume 356
Specifying storage for version 0 DCO plexes 357
Removing a version 0 DCO and DCO volume 358
Reattaching a version 0 DCO and DCO volume 359

Chapter 10 Creating and administering volume sets

Creating a volume set 362
Adding a volume to a volume set 362
Listing details of volume sets 363
Stopping and starting volume sets 363
Removing a volume from a volume set 364
Raw device node access to component volumes 364
Enabling raw device access when creating a volume set 365
Displaying the raw device access settings for a volume set 366
Controlling raw device access for an existing volume set 366

Chapter 11 Configuring off-host processing

Implementing off-host processing solutions 370
Implementing off-host online backup 371
Implementing decision support 374

Chapter 12 Administering hot-relocation

How hot-relocation works 380
Partial disk failure mail messages 383
Complete disk failure mail messages 384
How space is chosen for relocation 384
Configuring a system for hot-relocation 385
Displaying spare disk information 386
Marking a disk as a hot-relocation spare 387
Removing a disk from use as a hot-relocation spare 388
Excluding a disk from hot-relocation use 388
Making a disk available for hot-relocation use 389
Configuring hot-relocation to use only spare disks 390
Moving and unrelocating subdisks 390
Moving and unrelocating subdisks using vxdiskadm 391
Moving and unrelocating subdisks using vxassist 392
Moving and unrelocating subdisks using vxunreloc 392
Restarting vxunreloc after errors 394
Modifying the behavior of hot-relocation 395

Chapter 13 Administering cluster functionality

Overview of cluster volume management 398
Private and shared disk groups 401
Activation modes of shared disk groups 402
Connectivity policy of shared disk groups 404
Effect of disk connectivity on cluster reconfiguration 409
Limitations of shared disk groups 409
Cluster initialization and configuration 410
Cluster reconfiguration 410
Volume reconfiguration 413
Node shutdown 416
Node abort 417
Cluster shutdown 417
Multiple host failover configurations 417
Import lock 418
Failover 418
Corruption of disk group configuration 419
Administering VxVM in cluster environments 420
Requesting node status and discovering the master node 420
Determining if a disk is shareable 421
Listing shared disk groups 421
Creating a shared disk group 422
Importing disk groups as shared 423

Converting a disk group from shared to private 424
Moving objects between disk groups 424
Splitting disk groups 424
Joining disk groups 424
Changing the activation mode on a shared disk group 425
Setting the disk detach policy on a shared disk group 425
Setting the disk group failure policy on a shared disk group 426
Creating volumes with exclusive open access by a node 426
Setting exclusive open access to a volume by a node 426
Displaying the cluster protocol version 427
Displaying the supported cluster protocol version range 427
Upgrading the cluster protocol version 428
Recovering volumes in shared disk groups 428
Obtaining cluster performance statistics 428

Chapter 14 Administering sites and remote mirrors

Configuring sites for hosts and disks 434
Configuring site-based allocation on a disk group 434
Configuring site consistency on a disk group 435
Configuring site consistency on a volume 435
Setting the siteread policy on a volume 436
Site-based allocation of storage to volumes 436
Examples of storage allocation using sites 438
Making an existing disk group site consistent 439
Fire drill — testing the configuration 440
Simulating site failure 440
Recovery from simulated site failure 440
Automatic site reattachment 440
Failure scenarios and recovery procedures 441
Recovery from a loss of site connectivity 442
Recovery from host failure 442
Recovery from storage failure 442
Recovery from site failure 443

Chapter 15 Using Storage Expert

About Storage Expert 445
How Storage Expert works 446
Before using Storage Expert 446
Running Storage Expert 446
Discovering what a rule does 447
Displaying rule attributes and their default values 447

Running a rule 447
Identifying configuration problems using Storage Expert 449
Recovery time 449
Disk groups 451
Disk striping 453
Disk sparing and relocation management 454
Hardware failures 454
Rootability 454
System name 454
Rule definitions and attributes 456

Chapter 16 Performance monitoring and tuning

Performance guidelines 463
Data assignment 463
Striping 464
Mirroring 464
Combining mirroring and striping 465
RAID-5 465
Volume read policies 466
Performance monitoring 467
Setting performance priorities 467
Obtaining performance data 467
Using performance data 469
Tuning VxVM 472
General tuning guidelines 472
Tuning guidelines for large systems 473
Changing the values of tunables 474
Tunable parameters 475

Appendix A Commands summary

Online manual pages 507
Section 1M – administrative commands 507
Section 4 – file formats 510
Section 7 – device driver interfaces 510

Appendix B Configuring Veritas Volume Manager

Setup tasks after installation 511
Adding unsupported disk arrays as JBODs 512
Adding foreign devices 512
Adding disks to disk groups 512
Guidelines for configuring storage 513
Mirroring guidelines 514

Dirty region logging guidelines 515
Striping guidelines 515
RAID-5 guidelines 516
Hot-relocation guidelines 516
Accessing volume devices 518
Controlling VxVM's view of multipathed devices 518
Configuring cluster support 518
Configuring shared disk groups 519
Converting existing VxVM disk groups to shared disk groups 519
Reconfiguration tasks 520
Changing the name of the default disk group 520
Enabling or disabling enclosure-based naming 520

Glossary 521

Index 531

Chapter 1

Understanding Veritas Volume Manager

Veritas™ Volume Manager (VxVM) by Symantec is a storage management subsystem that allows you to manage physical disks as logical devices called volumes. A VxVM volume appears to applications and the operating system as a physical disk on which file systems, databases and other managed data objects can be configured.

VxVM provides easy-to-use online disk storage management for computing environments and Storage Area Network (SAN) environments. By supporting the Redundant Array of Independent Disks (RAID) model, VxVM can be configured to protect against disk and hardware failure, and to increase I/O throughput. Additionally, VxVM provides features that enhance fault tolerance and fast recovery from disk failure.

VxVM overcomes physical restrictions imposed by hardware disk devices by providing a logical volume management layer. This allows volumes to span multiple disks.

VxVM provides the tools to improve performance and ensure data availability and integrity. You can also use VxVM to dynamically configure disk storage while the system is active.

The following sections of this chapter explain fundamental concepts of VxVM:

- VxVM and the operating system
- How VxVM handles storage management
- Volume layouts in VxVM

The following sections introduce you to advanced features of VxVM:

- Online relayout
- Volume resynchronization
- Dirty region logging

- Volume snapshots
- FastResync
- Hot-relocation
- Volume sets

Further information on administering Veritas Volume Manager may be found in the following documents:

- *Veritas Storage Foundation Cross-Platform Data Sharing Administrator's Guide*
 Provides more information on using the Cross-platform Data Sharing (CDS) feature of Veritas Volume Manager, which allows you to move VxVM disks and objects between machines that are running under different operating systems.

 Note: The CDS feature requires a Veritas Storage Foundation license.

- *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide*
 Describes the command-line interface to the Veritas Intelligent Storage Provisioning (ISP) feature, which uses a rule-based engine to create VxVM objects and make optimal usage of the available storage.
- *Veritas FlashSnap Point-In-Time Copy Solutions Administrator's Guide*
 Provides guidelines on using the features of the FlashSnap software to implement various point-in-time copy solutions for backup, and database replication.

 Note: The FlashSnap feature requires a separate license.

- *Veritas Volume Manager Troubleshooting Guide*
 Describes recovery from hardware failure, disk group configuration and restoration, command and transaction logging, and common error messages together with suggested solutions.
- *Veritas Enterprise Administrator User's Guide*
 Describes how to use the Veritas Enterprise Administrator — the graphical user interface to Veritas Volume Manager. More detailed information is available in the VEA online help.

VxVM and the operating system

VxVM operates as a subsystem between your operating system and your data management systems, such as file systems and database management systems. VxVM is tightly coupled with the operating system. Before a disk can be brought under VxVM control, the disk must be accessible through the operating system device interface. VxVM is layered on top of the operating system interface services, and is dependent upon how the operating system accesses physical disks.

VxVM is dependent upon the operating system for the following functionality:

- operating system (disk) devices
- device handles
- VxVM dynamic multipathing (DMP) metadevice

This guide introduces you to the VxVM commands which are used to carry out the tasks associated with VxVM objects. These commands are described on the relevant manual pages and in the chapters of this guide where VxVM tasks are described.

VxVM relies on the following constantly-running daemons and kernel threads for its operation:

- `vxconfigd`—The VxVM configuration daemon maintains disk and group configurations and communicates configuration changes to the kernel, and modifies configuration information stored on disks.
- `vxiod`—VxVM I/O kernel threads provide extended I/O operations without blocking calling processes. By default, 16 I/O threads are started at boot time, and at least one I/O thread must continue to run at all times.
- `vxrelocd`—The hot-relocation daemon monitors VxVM for events that affect redundancy, and performs hot-relocation to restore redundancy.

How data is stored

There are several methods used to store data on physical disks. These methods organize data on the disk so the data can be stored and retrieved efficiently. The basic method of disk organization is called *formatting*. Formatting prepares the hard disk so that files can be written to and retrieved from the disk by using a prearranged storage pattern.

Hard disks are formatted, and information stored, using two methods: physical-storage layout and logical-storage layout. VxVM uses the *logical-storage layout* method. The types of storage layout supported by VxVM are introduced in this chapter.

How VxVM handles storage management

VxVM uses two types of *objects* to handle storage management: *physical objects* and *virtual objects*.

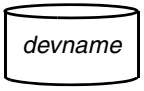
- Physical objects—physical disks or other hardware with block and raw operating system device interfaces that are used to store data.
- Virtual objects—When one or more physical disks are brought under the control of VxVM, it creates virtual objects called *volumes* on those physical disks. Each volume records and retrieves data from one or more physical disks. Volumes are accessed by file systems, databases, or other applications in the same way that physical disks are accessed. Volumes are also composed of other virtual objects (plexes and subdisks) that are used in changing the volume configuration. Volumes and their virtual components are called virtual objects or VxVM objects.

Physical objects—physical disks

A *physical disk* is the basic storage device (media) where the data is ultimately stored. You can access the data on a physical disk by using a *device name* to locate the disk.

Figure 1-1 shows how a physical disk and device name (*devname*) are illustrated in this document.

Figure 1-1 Physical disk example



In HP-UX 11i v3, disks may be identified either by their legacy device name, which takes the form `c#t#d#`, or by their persistent (or *agile*) device name, which takes the form `disk##`.

In a legacy device name, `c#` specifies the controller, `t#` specifies the target ID, and `d#` specifies the disk. For example, the device name `c0t0d0` is the entire hard disk that is connected to controller number `0` in the system, with a target ID of `0`, and physical disk number of `0`. The equivalent persistent device name might be `disk33`.

In this document, legacy device names are generally shown as this format is the same as the default format that is used by the Dynamic Discovery Layer (DDL) and Dynamic Multipathing (DMP) features of VxVM.

VxVM writes identification information on physical disks under VxVM control (VM disks). VxVM disks can be identified even after physical disk disconnection or system outages. VxVM can then re-form disk groups and logical objects to provide failure detection and to speed system recovery.

VxVM accesses all disks as entire physical disks without partitions.

Disk arrays

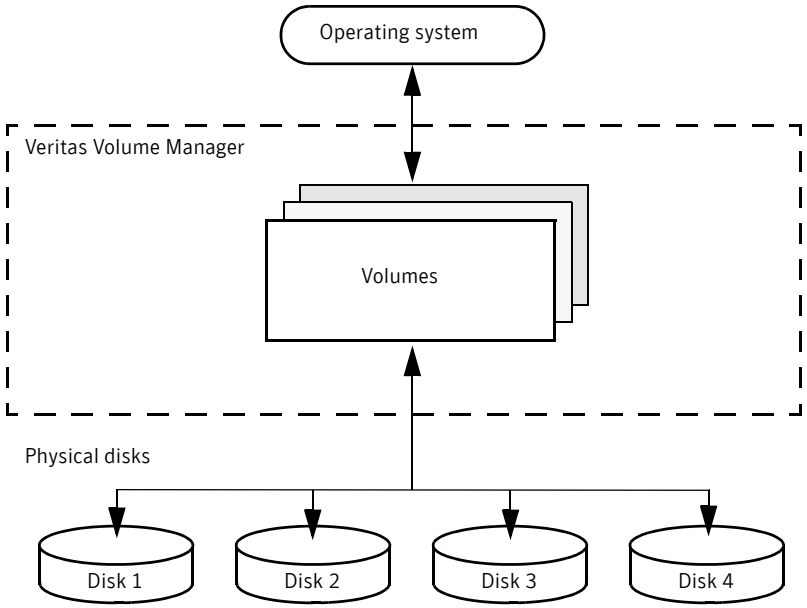
Performing I/O to disks is a relatively slow process because disks are physical devices that require time to move the heads to the correct position on the disk before reading or writing. If all of the read or write operations are done to individual disks, one at a time, the read-write time can become unmanageable. Performing these operations on multiple disks can help to reduce this problem.

A *disk array* is a collection of physical disks that VxVM can represent to the operating system as one or more virtual disks or *volumes*. The volumes created by VxVM look and act to the operating system like physical disks. Applications that interact with volumes should work in the same way as with physical disks.

Figure 1-2 illustrates how VxVM represents the disks in a disk array as several volumes to the operating system.

Data can be spread across several disks within an array to distribute or *balance* I/O operations across the disks. Using parallel I/O across multiple disks in this way improves I/O performance by increasing data transfer speed and overall throughput for the array.

Figure 1-2 How VxVM presents the disks in a disk array as volumes to the operating system



Multipathed disk arrays

Some disk arrays provide multiple ports to access their disk devices. These ports, coupled with the host bus adaptor (HBA) controller and any data bus or I/O processor local to the array, make up multiple hardware paths to access the disk devices. Such disk arrays are called *multipathed disk arrays*. This type of disk array can be connected to host systems in many different configurations, (such as multiple ports connected to different controllers on a single host, chaining of the ports through a single controller on a host, or ports connected to different hosts simultaneously).

HP-UX 11i v3 provides its own native multipathing solution in addition to the Dynamic Multipathing (DMP) feature that is used by VxVM. These two multipathing solutions can coexist on the same system. For more information, see "DMP coexistence with HP-UX native multipathing" on page 130.

Device discovery

Device discovery is the term used to describe the process of discovering the disks that are attached to a host. This feature is important for DMP because it needs to support a growing number of disk arrays from a number of vendors. In conjunction with the ability to discover the devices attached to a host, the

Device Discovery service enables you to add support dynamically for new disk arrays. This operation, which uses a facility called the Device Discovery Layer (DDL), is achieved without the need for a reboot.

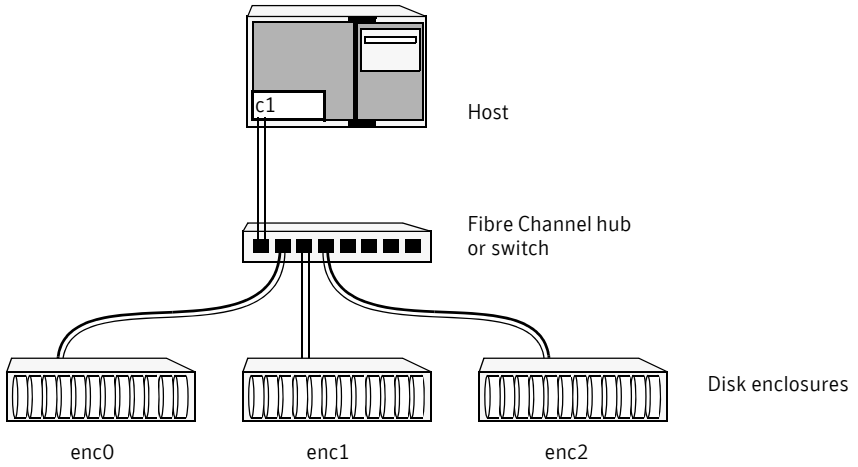
This means that you can dynamically add a new disk array to a host, and run a command which scans the operating system's device tree for all the attached disk devices, and reconfigures DMP with the new device database. For more information, see "Administering the Device Discovery Layer" on page 85.

Enclosure-based naming

Enclosure-based naming provides an alternative to the disk device naming described in "Physical objects—physical disks" on page 20. This allows disk devices to be named for enclosures rather than for the controllers through which they are accessed. In a Storage Area Network (SAN) that uses Fibre Channel hubs or fabric switches, information about disk location provided by the operating system may not correctly indicate the physical location of the disks. For example, c#t#d# naming assigns controller-based device names to disks in separate enclosures that are connected to the same host controller. Enclosure-based naming allows VxVM to access enclosures as separate physical entities. By configuring redundant copies of your data on separate enclosures, you can safeguard against failure of one or more enclosures.

In a typical SAN environment, host controllers are connected to multiple enclosures in a daisy chain or through a Fibre Channel hub or fabric switch as illustrated in Figure 1-3.

Figure 1-3 Example configuration for disk enclosures connected via a fibre channel hub or switch



In such a configuration, enclosure-based naming can be used to refer to each disk within an enclosure. For example, the device names for the disks in enclosure `enc0` are named `enc0_0`, `enc0_1`, and so on. The main benefit of this scheme is that it allows you to quickly determine where a disk is physically located in a large SAN configuration.

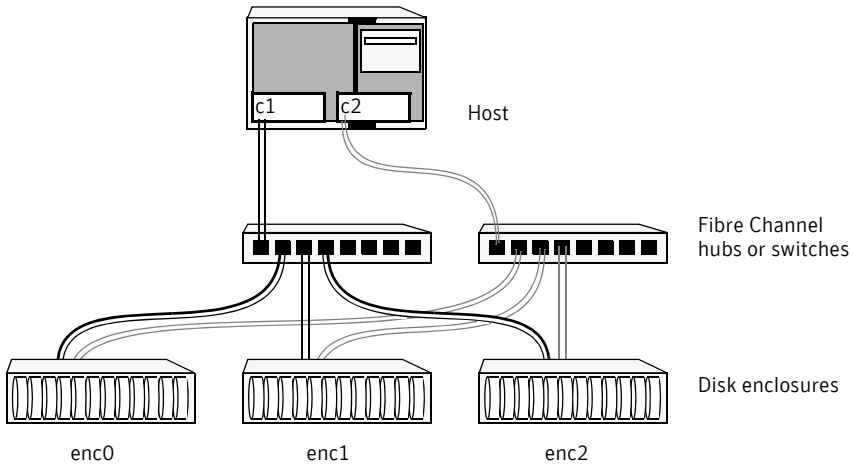
Note: In many advanced disk arrays, you can use hardware-based storage management to represent several physical disks as one logical disk device to the operating system. In such cases, VxVM also sees a single logical disk device rather than its component disks. For this reason, when reference is made to a *disk* within an enclosure, this disk may be either a physical or a logical device.

Another important benefit of enclosure-based naming is that it enables VxVM to avoid placing redundant copies of data in the same enclosure. This is a good thing to avoid as each enclosure can be considered to be a separate fault domain. For example, if a mirrored volume were configured only on the disks in enclosure `enc1`, the failure of the cable between the hub and the enclosure would make the entire volume unavailable.

If required, you can replace the default name that VxVM assigns to an enclosure with one that is more meaningful to your configuration. See “Renaming an enclosure” on page 155 for details.

In High Availability (HA) configurations, redundant-loop access to storage can be implemented by connecting independent controllers on the host to separate hubs with independent paths to the enclosures as shown in Figure 1-4.

Figure 1-4 Example HA configuration using multiple hubs or switches to provide redundant loop access



Such a configuration protects against the failure of one of the host controllers (`c1` and `c2`), or of the cable between the host and one of the hubs. In this example, each disk is known by the same name to VxVM for all of the paths over which it can be accessed. For example, the disk device `enc0_0` represents a single disk for which two different paths are known, such as `c1t99d0` and `c2t99d0`.

Note: The native multipathing feature of HP-UX 11i v3 similarly maps the various physical paths to a disk, and presents these as a single persistent device with a name of the form disk##. However, this mechanism is independent of that used by VxVM. For instructions on administering native multipathing with `Base-VxVM` and `VxVM-Full`, please consult the *Release Notes* for VxVM 5.0 on 11i v3.

To take account of fault domains when configuring data redundancy, you can control how mirrored volumes are laid out across enclosures as described in "Mirroring across targets, controllers or enclosures" on page 255.

See "Disk device naming in VxVM" on page 78 and "Changing the disk-naming scheme" on page 91 for details of the standard and the enclosure-based naming schemes, and how to switch between them.

Virtual objects

Virtual objects in VxVM include the following:

- Disk groups
- VM disks
- Subdisks
- Plexes
- Volumes

The connection between physical objects and VxVM objects is made when you place a physical disk under VxVM control.

After installing VxVM on a host system, you must bring the contents of physical disks under VxVM control by collecting the VM disks into disk groups and allocating the disk group space to create logical volumes.

Note: To bring the physical disk under VxVM control, the disk must not be under LVM control. For more information on how LVM and VM disks co-exist or how to convert LVM disks to VM disks, see the *Veritas Volume Manager Migration Guide*

Bringing the contents of physical disks under VxVM control is accomplished only if VxVM takes control of the physical disks and the disk is not under control of another storage manager.

VxVM creates *virtual objects* and makes logical connections between the objects. The virtual objects are then used by VxVM to do storage management tasks.

Note: The `vxprint` command displays detailed information on existing VxVM objects. For additional information on the `vxprint` command, see "Displaying volume information" on page 264 and the `vxprint`(1M) manual page.

Combining virtual objects in VxVM

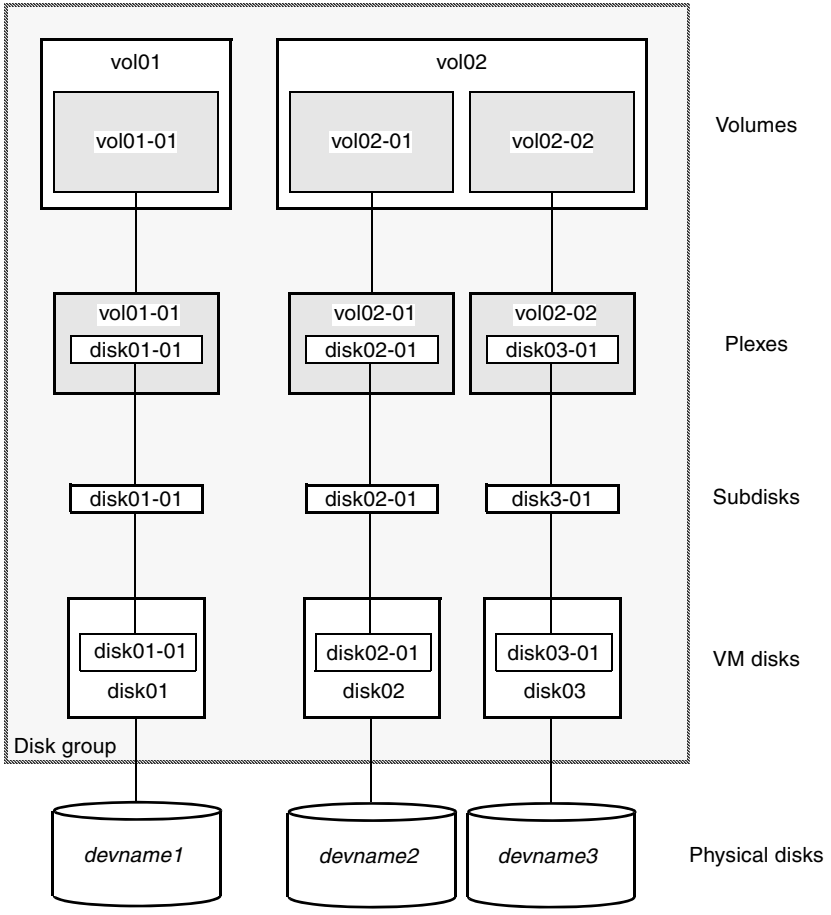
VxVM virtual objects are combined to build volumes. The virtual objects contained in volumes are VM disks, disk groups, subdisks, and plexes. Veritas Volume Manager objects are organized as follows:

- VM disks are grouped into disk groups

- Subdisks (each representing a specific region of a disk) are combined to form plexes
- Volumes are composed of one or more plexes

Figure 1-5 shows the connections between Veritas Volume Manager virtual objects and how they relate to physical disks. The disk group contains three VM disks which are used to create two volumes. Volume `vol01` is simple and has a single plex. Volume `vol02` is a mirrored volume with two plexes.

Figure 1-5 Connection between objects in VxVM



The various types of virtual objects (disk groups, VM disks, subdisks, plexes and volumes) are described in the following sections. Other types of objects exist in

Veritas Volume Manager, such as data change objects (DCOs), and cache objects, to provide extended functionality. These objects are discussed later in this chapter.

Disk groups

A *disk group* is a collection of disks that share a common configuration, and which are managed by VxVM (see "VM disks" on page 28). A disk group configuration is a set of records with detailed information about related VxVM objects, their attributes, and their connections. A disk group name can be up to 31 characters long.

In releases prior to VxVM 4.0, the default disk group was `rootdg` (the *root disk group*). For VxVM to function, the `rootdg` disk group had to exist and it had to contain at least one disk. This requirement no longer exists, and VxVM can work without any disk groups configured (although you must set up at least one disk group before you can create any volumes of otherVxVM objects). For more information about changes to disk group configuration, see "Creating and administering disk groups" on page 165.

You can create additional disk groups when you need them. Disk groups allow you to group disks into logical collections. A disk group and its components can be moved as a unit from one host machine to another. The ability to move whole volumes and disks between disk groups, to split whole volumes and disks between disk groups, and to join disk groups is described in "Reorganizing the contents of disk groups" on page 195.

Volumes are created within a disk group. A given volume and its plexes and subdisks must be configured from disks in the same disk group.

VM disks

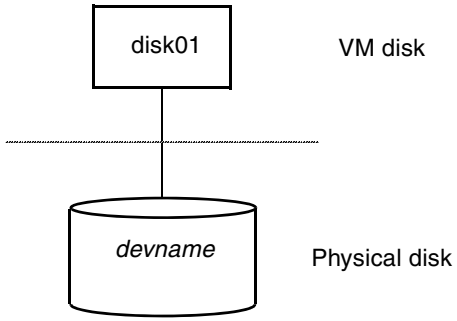
When you place a physical disk under VxVM control, a VM disk is assigned to the physical disk. A VM disk is under VxVM control and is usually in a disk group. Each VM disk corresponds to one physical disk. VxVM allocates storage from a contiguous area of VxVM disk space.

A VM disk typically includes a *public region* (allocated storage) and a small *private region* where VxVM internal configuration information is stored.

Each VM disk has a unique *disk media name* (a virtual disk name). You can either define a disk name of up to 31 characters, or allow VxVM to assign a default name that takes the form *`diskgroup`*`##`, where *diskgroup* is the name of the disk group to which the disk belongs (see "Disk groups" on page 28).

Figure 1-6 shows a VM disk with a media name of `disk01` that is assigned to the physical disk *devname*.

Figure 1-6 VM disk example



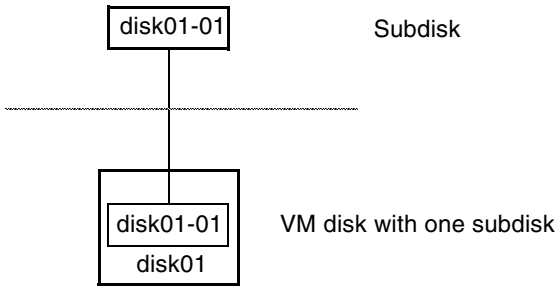
Subdisks

A *subdisk* is a set of contiguous disk blocks. A block is a unit of space on the disk. VxVM allocates disk space using subdisks. A VM disk can be divided into one or more subdisks. Each subdisk represents a specific portion of a VM disk, which is mapped to a specific region of a physical disk.

The default name for a VM disk is `diskgroup##` and the default name for a subdisk is `diskgroup##-##`, where *diskgroup* is the name of the disk group to which the disk belongs (see "Disk groups" on page 28).

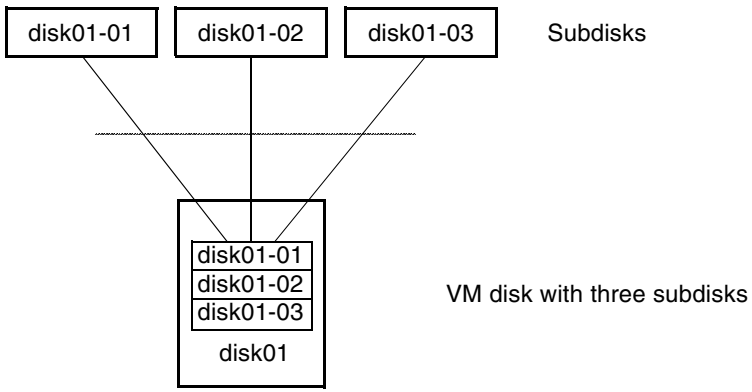
In Figure 1-7, `disk01-01` is the name of the first subdisk on the VM disk named `disk01`.

Figure 1-7 Subdisk example



A VM disk can contain multiple subdisks, but subdisks cannot overlap or share the same portions of a VM disk. Figure 1-8 shows a VM disk with three subdisks. (The VM disk is assigned to one physical disk.)

Figure 1-8 Example of three subdisks assigned to one VM Disk



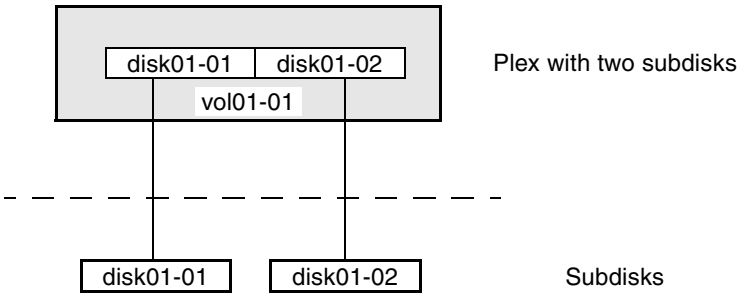
Any VM disk space that is not part of a subdisk is free space. You can use free space to create new subdisks.

VxVM release 3.0 or higher supports the concept of layered volumes in which subdisks can contain volumes. For more information, see “Layered volumes” on page 51.

Plexes

VxVM uses subdisks to build virtual objects called *plexes*. A plex consists of one or more subdisks located on one or more physical disks. For example, see the plex `vol01-01` shown in Figure 1-9.

Figure 1-9 Example of a plex with two subdisks



You can organize data on subdisks to form a plex by using the following methods:

- concatenation
- striping (RAID-0)
- mirroring (RAID-1)
- striping with parity (RAID-5)

Concatenation, striping (RAID-0), mirroring (RAID-1) and RAID-5 are described in "Volume layouts in VxVM" on page 34.

Volumes

A *volume* is a virtual disk device that appears to applications, databases, and file systems like a physical disk device, but does not have the physical limitations of a physical disk device. A volume consists of one or more plexes, each holding a copy of the selected data in the volume. Due to its virtual nature, a volume is not restricted to a particular disk or a specific area of a disk. The configuration of a volume can be changed by using VxVM user interfaces. Configuration changes can be accomplished without causing disruption to applications or file systems that are using the volume. For example, a volume can be mirrored on separate disks or moved to use different disk storage.

Note: VxVM uses the default naming conventions of `vol##` for volumes and `vol##-##` for plexes in a volume. For ease of administration, you can choose to select more meaningful names for the volumes that you create.

A volume may be created under the following constraints:

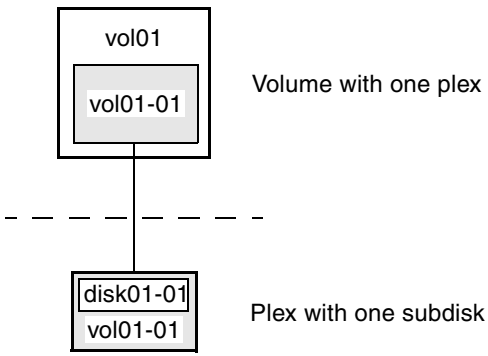
- Its name can contain up to 31 characters.
- It can consist of up to 32 plexes, each of which contains one or more subdisks.
- It must have at least one associated plex that has a complete copy of the data in the volume with at least one associated subdisk.
- All subdisks within a volume must belong to the same disk group.

Note: You can use the Veritas Intelligent Storage Provisioning (ISP) feature to create and administer application volumes. These volumes are very similar to the traditional VxVM volumes that are described in this chapter. However, there are significant differences between the functionality of the two types of volume that prevent them from being used interchangeably. Refer to the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for more information about creating and administering ISP application volumes.

In Figure 1-10, volume `vol01` has the following characteristics:

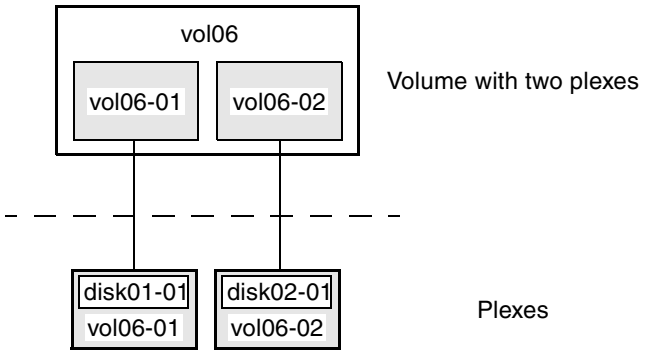
- It contains one plex named `vol01-01`.
- The plex contains one subdisk named `disk01-01`.
- The subdisk `disk01-01` is allocated from VM disk `disk01`.

Figure 1-10 Example of a volume with one plex



In Figure 1-11 a volume, `vol06`, with two data plexes is *mirrored*. Each plex of the mirror contains a complete copy of the volume data.

Figure 1-11 Example of a volume with two plexes



Volume `vol06` has the following characteristics:

- It contains two plexes named `vol06-01` and `vol06-02`.
- Each plex contains one subdisk.
- Each subdisk is allocated from a different VM disk (`disk01` and `disk02`).

For more information, see "Mirroring (RAID-1)" on page 42.

Volume layouts in VxVM

A VxVM virtual device is defined by a volume. A volume has a layout defined by the association of a volume to one or more plexes, each of which map to subdisks. The volume presents a virtual device interface that is exposed to other applications for data access. These logical building blocks re-map the volume address space through which I/O is re-directed at run-time.

Different volume layouts each provide different levels of storage service. A volume layout can be configured and reconfigured to match particular levels of desired storage service.

Implementation of non-layered volumes

In a *non-layered* volume, a subdisk is restricted to mapping directly to a VM disk. This allows the subdisk to define a contiguous extent of storage space backed by the public region of a VM disk. When active, the VM disk is directly associated with an underlying physical disk. The combination of a volume layout and the physical disks therefore determines the storage service available from a given virtual device.

Implementation of layered volumes

A *layered* volume is constructed by mapping its subdisks to underlying volumes. The subdisks in the underlying volumes must map to VM disks, and hence to attached physical storage.

Layered volumes allow for more combinations of logical compositions, some of which may be desirable for configuring a virtual device. Because permitting free use of layered volumes throughout the command level would have resulted in unwieldy administration, some ready-made layered volume configurations are designed into VxVM.

See “Layered volumes” on page 51.

These ready-made configurations operate with built-in rules to automatically match desired levels of service within specified constraints. The automatic configuration is done on a “best-effort” basis for the current command invocation working against the current configuration.

To achieve the desired storage service from a set of virtual devices, it may be necessary to include an appropriate set of VM disks into a disk group, and to execute multiple configuration commands.

To the extent that it can, VxVM handles initial configuration and on-line re-configuration with its set of layouts and administration interface to make this job easier and more deterministic.

Layout methods

Data in virtual objects is organized to create volumes by using the following layout methods:

- Concatenation and spanning
- Striping (RAID-0)
- Mirroring (RAID-1)
- Striping plus mirroring (mirrored-stripe or RAID-0+1)
- Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)
- RAID-5 (striping with parity)

The following sections describe each layout method.

Concatenation and spanning

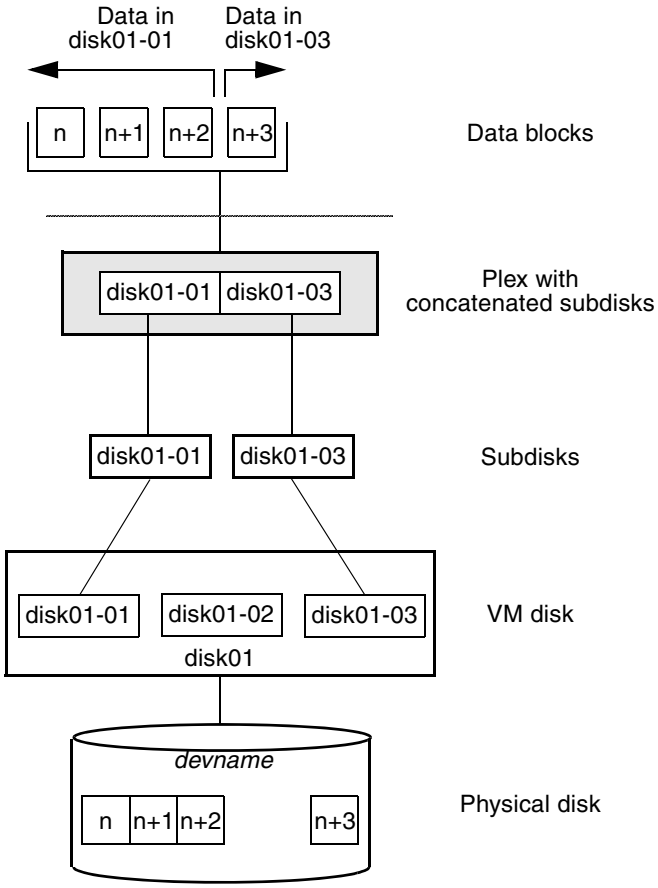
Concatenation maps data in a linear manner onto one or more subdisks in a plex. To access all of the data in a concatenated plex sequentially, data is first accessed in the first subdisk from beginning to end. Data is then accessed in the remaining subdisks sequentially from beginning to end, until the end of the last subdisk.

The subdisks in a concatenated plex do not have to be physically contiguous and can belong to more than one VM disk. Concatenation using subdisks that reside on more than one VM disk is called *spanning*.

Figure 1-12 shows the concatenation of two subdisks from the same VM disk. The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks on the same physical disk.

The remaining free space in the subdisk, `disk01-02`, on VM disk, `disk01`, can be put to other uses.

Figure 1-12 Example of concatenation

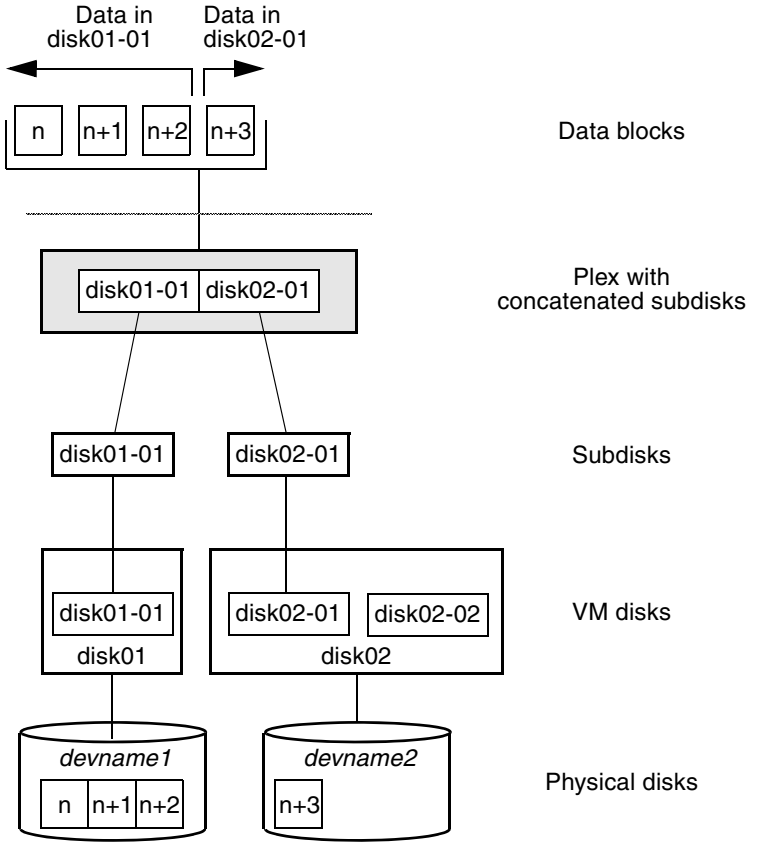


You can use concatenation with multiple subdisks when there is insufficient contiguous space for the plex on any one disk. This form of concatenation can be used for load balancing between disks, and for head movement optimization on a particular disk.

Figure 1-13 shows data spread over two subdisks in a spanned plex. The blocks `n`, `n+1`, `n+2` and `n+3` (numbered relative to the start of the plex) are contiguous on the plex, but actually come from two distinct subdisks from two distinct physical disks.

The remaining free space in the subdisk `disk02-02` on VM disk `disk02` can be put to other uses.

Figure 1-13 Example of spanning



Caution: Spanning a plex across multiple disks increases the chance that a disk failure results in failure of the assigned volume. Use mirroring or RAID-5 (both described later) to reduce the risk that a single disk failure results in a volume failure.

See "Creating a volume on any disk" on page 243 for information on how to create a concatenated volume that may span several disks.

Striping (RAID-0)

Note: You need a full license to use this feature.

Striping (RAID-0) is useful if you need large amounts of data written to or read from physical disks, and performance is important. Striping is also helpful in balancing the I/O load from multi-user applications across multiple disks. By using parallel data transfer to and from multiple disks, striping significantly improves data-access performance.

Striping maps data so that the data is interleaved among two or more physical disks. A striped plex contains two or more subdisks, spread out over two or more physical disks. Data is allocated alternately and evenly to the subdisks of a striped plex.

The subdisks are grouped into “columns,” with each physical disk limited to one column. Each column contains one or more subdisks and can be derived from one or more physical disks. The number and sizes of subdisks per column can vary. Additional subdisks can be added to columns, as necessary.

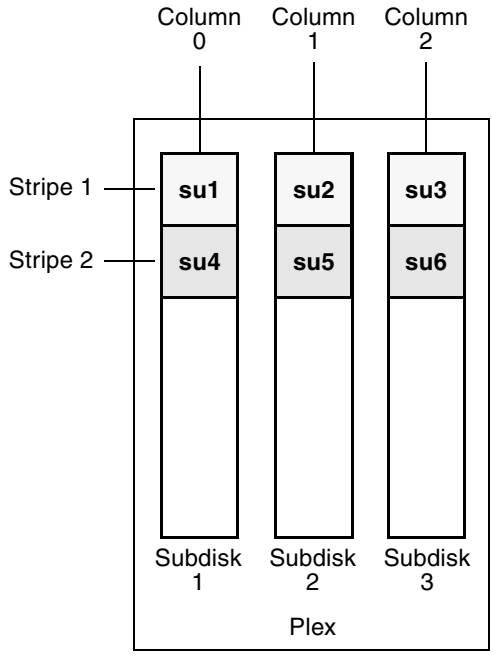
Caution: Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure will result in failure of that volume.

If five volumes are striped across the same five disks, then failure of any one of the five disks will require that all five volumes be restored from a backup. If each volume is on a separate disk, only one volume has to be restored. (As an alternative to striping, use mirroring or RAID-5 to substantially reduce the chance that a single disk failure results in failure of a large number of volumes.)

Data is allocated in equal-sized units (*stripe units*) that are interleaved between the columns. Each stripe unit is a set of contiguous blocks on a disk. The default stripe unit size (or *width*) is 64 kilobytes.

For example, if there are three columns in a striped plex and six stripe units, data is striped over the three columns, as illustrated in Figure 1-14.

Figure 1-14 Striping across three columns



A *stripe* consists of the set of stripe units at the same positions across all columns. In the figure, stripe units 1, 2, and 3 constitute a single stripe.

Viewed in sequence, the first stripe consists of:

- stripe unit 1 in column 0
- stripe unit 2 in column 1
- stripe unit 3 in column 2

The second stripe consists of:

- stripe unit 4 in column 0
- stripe unit 5 in column 1
- stripe unit 6 in column 2

Striping continues for the length of the columns (if all columns are the same length), or until the end of the shortest column is reached. Any space remaining at the end of subdisks in longer columns becomes unused space.

Figure 1-15 shows a striped plex with three equal sized, single-subdisk columns. There is one column per physical disk. This example shows three subdisks that occupy all of the space on the VM disks. It is also possible for each subdisk in a striped plex to occupy only a portion of the VM disk, which leaves free space for other disk management tasks.

Figure 1-15 Example of a striped plex with one subdisk per column

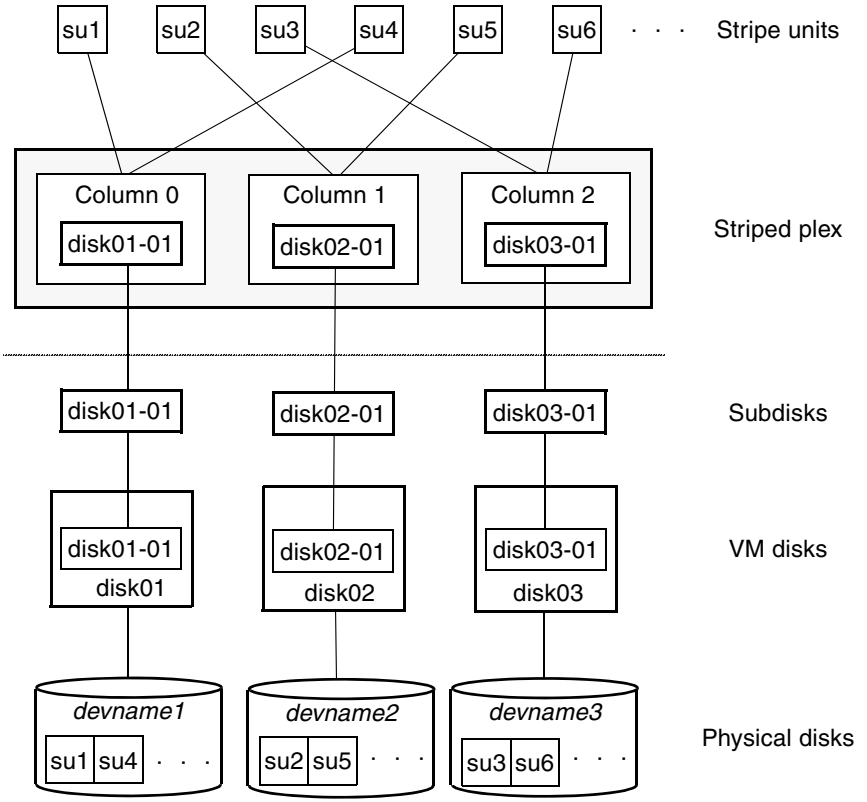
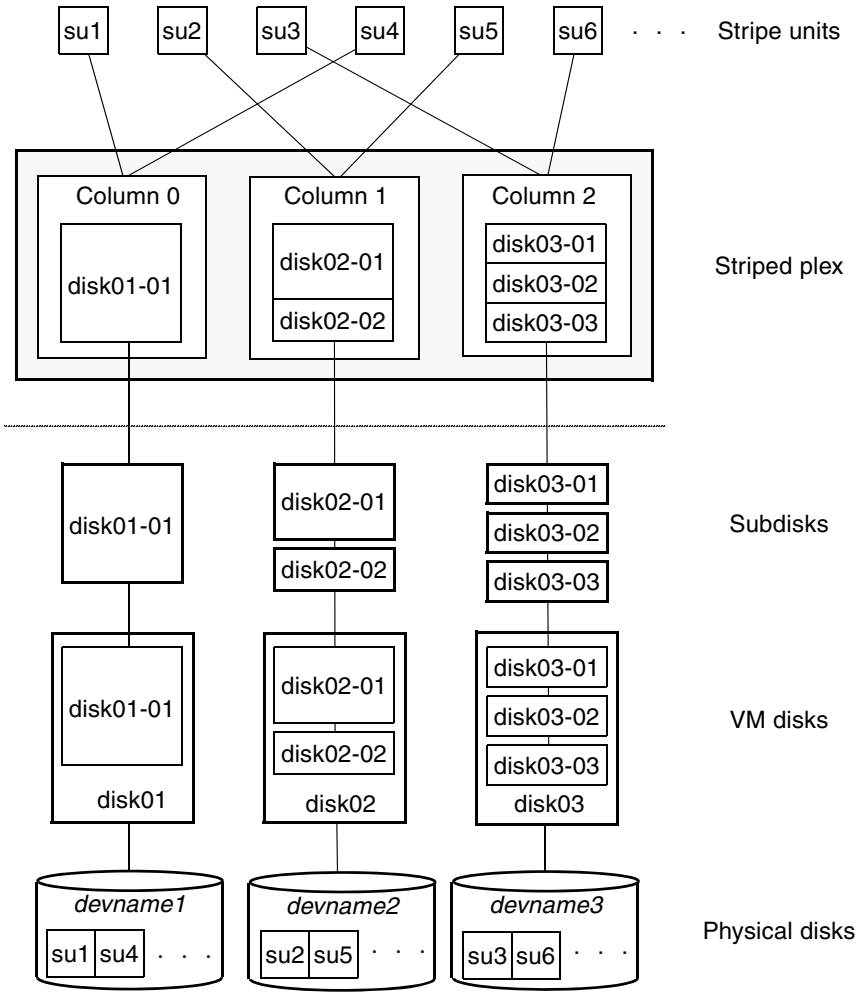


Figure 1-16 illustrates a striped plex with three columns containing subdisks of different sizes. Each column contains a different number of subdisks. There is one column per physical disk. Striped plexes can be created by using a single subdisk from each of the VM disks being striped across. It is also possible to allocate space from different regions of the same disk or from another disk (for example, if the size of the plex is increased). Columns can also contain subdisks from different VM disks.

Figure 1-16 Example of a striped plex with concatenated subdisks per column



See “Creating a striped volume” on page 253 for information on how to create a striped volume.

Mirroring (RAID-1)

Note: You need a full license to use this feature with disks other than the root disk.

Mirroring uses multiple mirrors (plexes) to duplicate the information contained in a volume. In the event of a physical disk failure, the plex on the failed disk becomes unavailable, but the system continues to operate using the unaffected mirrors.

Note: Although a volume can have a single plex, at least two plexes are required to provide redundancy of data. Each of these plexes must contain disk space from *different* disks to achieve redundancy.

When striping or spanning across a large number of disks, failure of any one of those disks can make the entire plex unusable. Because the likelihood of one out of several disks failing is reasonably high, you should consider mirroring to improve the reliability (and availability) of a striped or spanned volume.

See "Creating a mirrored volume" on page 249 for information on how to create a mirrored volume.

Disk duplexing, in which each mirror exists on a separate controller, is also supported. See "Mirroring across targets, controllers or enclosures" on page 255 for details.

Striping plus mirroring (mirrored-stripe or RAID-0+1)

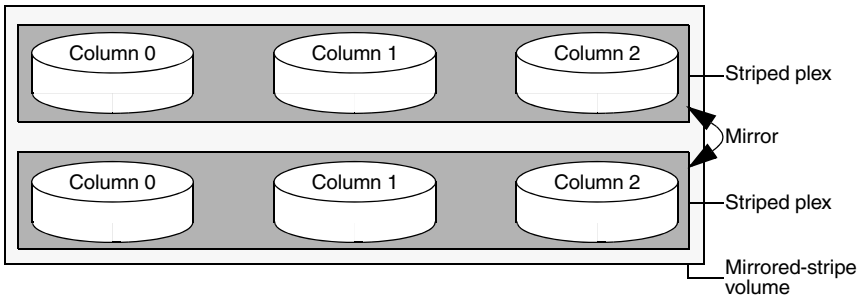
Note: You need a full license to use this feature.

VxVM supports the combination of mirroring above striping. The combined layout is called a *mirrored-stripe* layout. A mirrored-stripe layout offers the dual benefits of striping to spread data across multiple disks, while mirroring provides redundancy of data.

For mirroring above striping to be effective, the striped plex and its mirrors must be allocated from *separate* disks.

Figure 1-17 shows an example where two plexes, each striped across three disks, are attached as mirrors to the same volume to create a mirrored-stripe volume.

Figure 1-17 Mirrored-stripe volume laid out on six disks



See "Creating a mirrored-stripe volume" on page 254 for information on how to create a mirrored-stripe volume.

The layout type of the data plexes in a mirror can be concatenated or striped. Even if only one is striped, the volume is still termed a mirrored-stripe volume. If they are all concatenated, the volume is termed a *mirrored-concatenated* volume.

Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)

Note: You need a full license to use this feature.

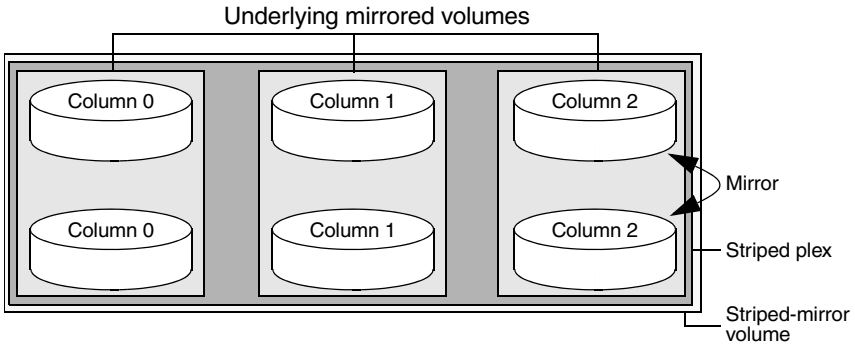
VxVM supports the combination of striping above mirroring. This combined layout is called a *striped-mirror* layout. Putting mirroring below striping mirrors each column of the stripe. If there are multiple subdisks per column, each subdisk can be mirrored individually instead of each column.

Note: A striped-mirror volume is an example of a layered volume. See "Layered volumes" on page 51 for more information.

As for a mirrored-stripe volume, a striped-mirror volume offers the dual benefits of striping to spread data across multiple disks, while mirroring provides redundancy of data. In addition, it enhances redundancy, and reduces recovery time after disk failure.

Figure 1-18 shows an example where a striped-mirror volume is created by using each of three existing 2-disk mirrored volumes to form a separate column within a striped plex.

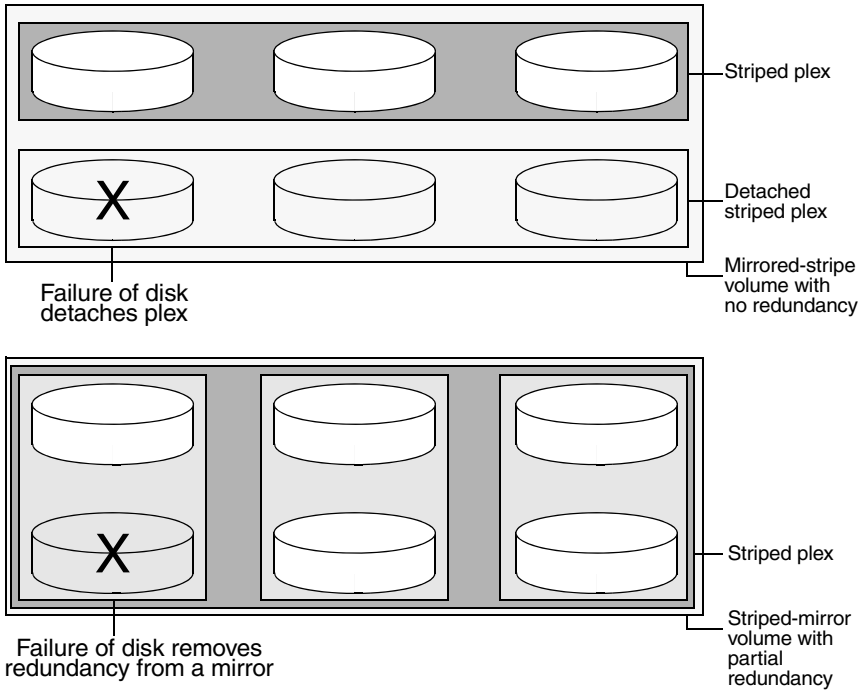
Figure 1-18 Striped-mirror volume laid out on six disks



See “Creating a striped-mirror volume” on page 254 for information on how to create a striped-mirrored volume.

As shown in Figure 1-19, the failure of a disk in a mirrored- stripe layout detaches an entire data plex, thereby losing redundancy on the entire volume. When the disk is replaced, the entire plex must be brought up to date. Recovering the entire plex can take a substantial amount of time. If a disk fails in a striped-mirror layout, only the failing subdisk must be detached, and only that portion of the volume loses redundancy. When the disk is replaced, only a portion of the volume needs to be recovered. Additionally, a mirrored-stripe volume is more vulnerable to being put out of use altogether should a second disk fail before the first failed disk has been replaced, either manually or by hot-relocation.

Figure 1-19 How the failure of a single disk affects mirrored-stripe and striped-mirror volumes



Compared to mirrored-stripe volumes, striped-mirror volumes are more tolerant of disk failure, and recovery time is shorter.

If the layered volume concatenates instead of striping the underlying mirrored volumes, the volume is termed a *concatenated-mirror* volume.

RAID-5 (striping with parity)

Note: VxVM supports RAID-5 for private disk groups, but not for shareable disk groups in a cluster environment. In addition, VxVM does not support the mirroring of RAID-5 volumes that are configured using Veritas Volume Manager software. Disk devices that support RAID-5 in hardware may be mirrored.

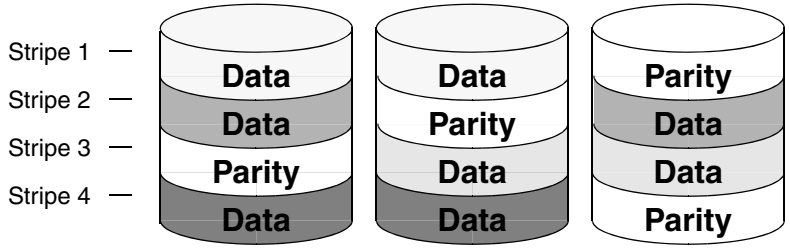
You need a full license to use this feature.

Although both mirroring (RAID-1) and RAID-5 provide redundancy of data, they use different methods. Mirroring provides data redundancy by maintaining multiple complete copies of the data in a volume. Data being written to a mirrored volume is reflected in all copies. If a portion of a mirrored volume fails, the system continues to use the other copies of the data.

RAID-5 provides data redundancy by using *parity*. Parity is a calculated value used to reconstruct data after a failure. While data is being written to a RAID-5 volume, parity is calculated by doing an exclusive OR (XOR) procedure on the data. The resulting parity is then written to the volume. The data and calculated parity are contained in a plex that is "striped" across multiple disks. If a portion of a RAID-5 volume fails, the data that was on that portion of the failed volume can be recreated from the remaining data and parity information. It is also possible to mix concatenation and striping in the layout.

Figure 1-20 shows parity locations in a RAID-5 array configuration. Every stripe has a column containing a parity stripe unit and columns containing data. The parity is spread over all of the disks in the array, reducing the write time for large independent writes because the writes do not have to wait until a single parity disk can accept the data.

Figure 1-20 Parity locations in a RAID-5 mode



RAID-5 volumes can additionally perform logging to minimize recovery time. RAID-5 volumes use RAID-5 logs to keep a copy of the data and parity currently being written. RAID-5 logging is optional and can be created along with RAID-5 volumes or added later.

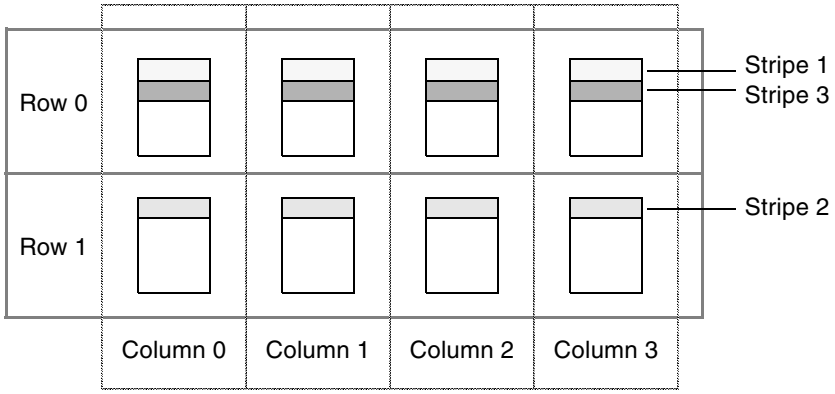
The implementation of RAID-5 in VxVM is described in "Veritas Volume Manager RAID-5 arrays" on page 47.

Traditional RAID-5 arrays

A *traditional* RAID-5 array is several disks organized in rows and columns. A *column* is a number of disks located in the same ordinal position in the array. A *row* is the minimal number of disks necessary to support the full width of a

parity stripe. Figure 1-21 shows the row and column arrangement of a traditional RAID-5 array.

Figure 1-21 Traditional RAID-5 array



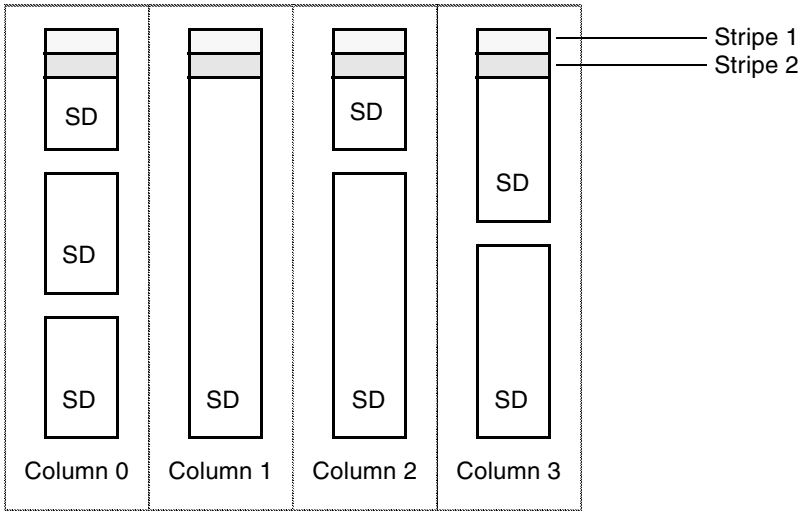
This traditional array structure supports growth by adding more rows per column. Striping is accomplished by applying the first stripe across the disks in Row 0, then the second stripe across the disks in Row 1, then the third stripe across the Row 0 disks, and so on. This type of array requires all disks columns, and rows to be of equal size.

Veritas Volume Manager RAID-5 arrays

The RAID-5 array structure in Veritas Volume Manager differs from the traditional structure. Due to the virtual nature of its disks and other objects, VxVM does not use rows. Instead, VxVM uses columns consisting of variable length subdisks as shown in Figure 1-22. Each subdisk represents a specific area of a disk.

VxVM allows each column of a RAID-5 plex to consist of a different number of subdisks. The subdisks in a given column can be derived from different physical disks. Additional subdisks can be added to the columns as necessary. Striping is implemented by applying the first stripe across each subdisk at the top of each column, then applying another stripe below that, and so on for the length of the columns. Equal-sized stripe units are used for each column. For RAID-5, the default stripe unit size is 16 kilobytes. See “Striping (RAID-0)” on page 38 for further information about stripe units.

Figure 1-22 Veritas Volume Manager RAID-5 array



Note: Mirroring of RAID-5 volumes is not supported.

See "Creating a RAID-5 volume" on page 256 for information on how to create a RAID-5 volume.

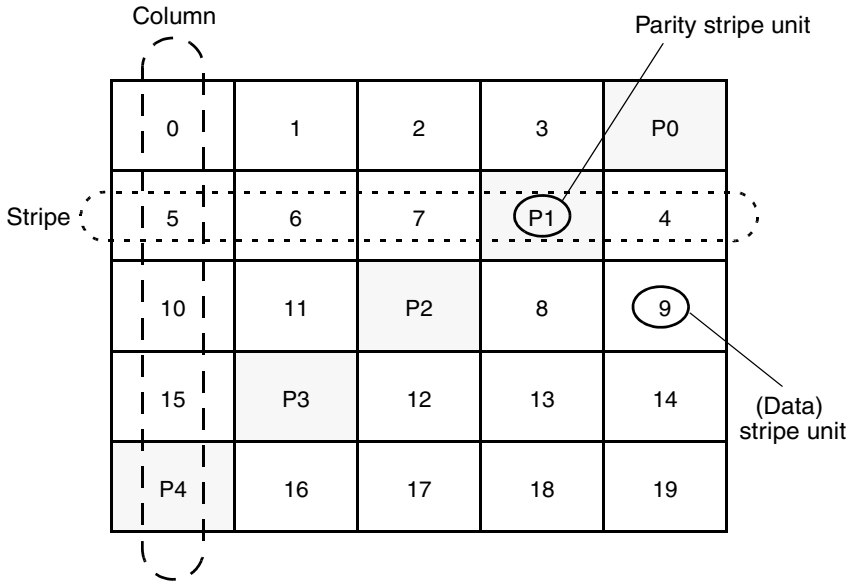
Left-symmetric layout

There are several layouts for data and parity that can be used in the setup of a RAID-5 array. The implementation of RAID-5 in VxVM uses a left-symmetric layout. This provides optimal performance for both random I/O operations and large sequential I/O operations. However, the layout selection is not as critical for performance as are the number of columns and the stripe unit size.

Left-symmetric layout stripes both data and parity across columns, placing the parity in a different column for every stripe of data. The first parity stripe unit is located in the rightmost column of the first stripe. Each successive parity stripe unit is located in the next stripe, shifted left one column from the previous parity stripe unit location. If there are more stripes than columns, the parity stripe unit placement begins in the rightmost column again.

Figure 1-23 shows a left-symmetric parity layout with five disks (one per column).

Figure 1-23 Left-symmetric layout



For each stripe, data is organized starting to the right of the parity stripe unit. In the figure, data organization for the first stripe begins at P0 and continues to stripe units 0-3. Data organization for the second stripe begins at P1, then continues to stripe unit 4, and on to stripe units 5-7. Data organization proceeds in this manner for the remaining stripes.

Each parity stripe unit contains the result of an exclusive OR (XOR) operation performed on the data in the data stripe units within the same stripe. If one column's data is inaccessible due to hardware or software failure, the data for each stripe can be restored by XORing the contents of the remaining columns data stripe units against their respective parity stripe units.

For example, if a disk corresponding to the whole or part of the far left column fails, the volume is placed in a degraded mode. While in degraded mode, the data from the failed column can be recreated by XORing stripe units 1-3 against parity stripe unit P0 to recreate stripe unit 0, then XORing stripe units 4, 6, and 7 against parity stripe unit P1 to recreate stripe unit 5, and so on.

Note: Failure of more than one column in a RAID-5 plex detaches the volume. The volume is no longer allowed to satisfy read or write requests. Once the failed columns have been recovered, it may be necessary to recover user data from backups.

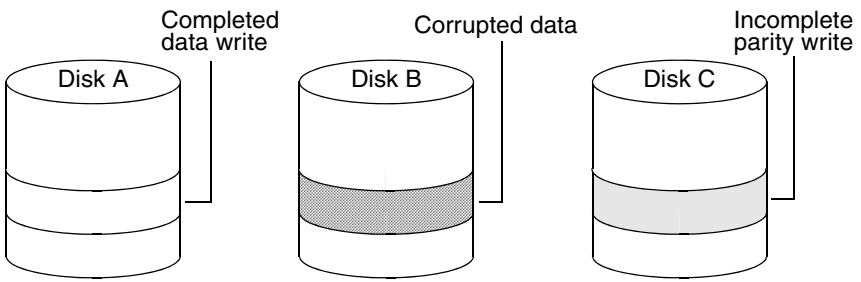
RAID-5 logging

Logging is used to prevent corruption of data during recovery by immediately recording changes to data and parity to a log area on a *persistent* device such as a volume on disk or in non-volatile RAM. The new data and parity are then written to the disks.

Without logging, it is possible for data not involved in any active writes to be lost or silently corrupted if both a disk in a RAID-5 volume and the system fail. If this double-failure occurs, there is no way of knowing if the data being written to the data portions of the disks or the parity being written to the parity portions have actually been written. Therefore, the recovery of the corrupted disk may be corrupted itself.

Figure 1-24 illustrates a RAID-5 volume configured across three disks (A, B and C). In this volume, recovery of disk B's corrupted data depends on disk A's data and disk C's parity both being complete. However, only the data write to disk A is complete. The parity write to disk C is incomplete, which would cause the data on disk B to be reconstructed incorrectly.

Figure 1-24 Incomplete write to a RAID-5 volume



This failure can be avoided by logging all data and parity writes before committing them to the array. In this way, the log can be replayed, causing the data and parity updates to be completed before the reconstruction of the failed drive takes place.

Logs are associated with a RAID-5 volume by being attached as log plexes. More than one log plex can exist for each RAID-5 volume, in which case the log areas are mirrored.

See “Adding a RAID-5 log” on page 283 for information on how to add a RAID-5 log to a RAID-5 volume.

Layered volumes

A *layered volume* is a virtual Veritas Volume Manager object that is built on top of other volumes. The layered volume structure tolerates failure better and has greater redundancy than the standard volume structure. For example, in a striped-mirror layered volume, each mirror (plex) covers a smaller area of storage space, so recovery is quicker than with a standard mirrored volume.

Figure 1-25 Example of a striped-mirror layered volume

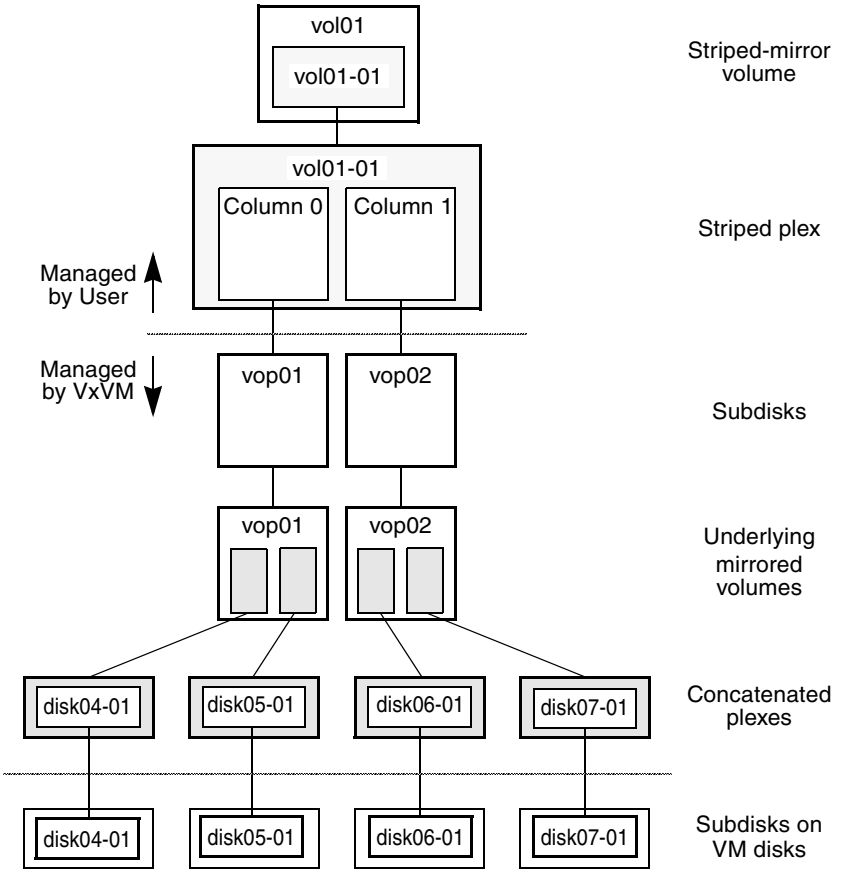


Figure 1-25 illustrates the structure of a typical layered volume. It shows subdisks with two columns, built on underlying volumes with each volume internally mirrored. The volume and striped plex in the “Managed by User” area allow you to perform normal tasks in VxVM. User tasks can be performed only on the top-level volume of a layered volume.

Underlying volumes in the “Managed by VxVM” area are used exclusively by VxVM and are not designed for user manipulation. You cannot detach a layered volume or perform any other operation on the underlying volumes by manipulating the internal structure. You can perform all necessary operations in the “Managed by User” area that includes the top-level volume and striped

plex (for example, resizing the volume, changing the column width, or adding a column).

System administrators can manipulate the layered volume structure for troubleshooting or other operations (for example, to place data on specific disks). Layered volumes are used by VxVM to perform the following tasks and operations:

- Creating striped-mirrors. (See “Creating a striped-mirror volume” on page 254, and the `vxassist`(1M) manual page.)
- Creating concatenated-mirrors. (See “Creating a concatenated-mirror volume” on page 249, and the `vxassist`(1M) manual page.)
- Online Relayout. (See “Online relayout” on page 54, and the `vxrelayout`(1M) and `vxassist`(1M) manual pages.)
- RAID-5 subdisk moves. (See the `vxsd`(1M) manual page.)
- Snapshots. (See “Administering volume snapshots” on page 303, and the `vxsnap`(1M) and `vxassist`(1M) manual pages.)

Online relayout

Note: You need a full license to use this feature.

Online relayout allows you to convert between storage layouts in VxVM, with uninterrupted data access. Typically, you would do this to change the redundancy or performance characteristics of a volume. VxVM adds redundancy to storage either by duplicating the data (mirroring) or by adding parity (RAID-5). Performance characteristics of storage in VxVM can be changed by changing the striping parameters, which are the number of columns and the stripe width.

See "Performing online relayout" on page 294 for details of how to perform online relayout of volumes in VxVM. Also see "Converting between layered and non-layered volumes" on page 300 for information about the additional volume conversion operations that are possible.

How online relayout works

Online relayout allows you to change the storage layouts that you have already created in place without disturbing data access. You can change the performance characteristics of a particular layout to suit your changed requirements. You can transform one layout to another by invoking a single command.

For example, if a striped layout with a 128KB stripe unit size is not providing optimal performance, you can use relayout to change the stripe unit size.

File systems mounted on the volumes do not need to be unmounted to achieve this transformation provided that the file system (such as Veritas File System) supports online shrink and grow operations.

Online relayout reuses the existing storage space and has space allocation policies to address the needs of the new layout. The layout transformation process converts a given volume to the destination layout by using minimal temporary space that is available in the disk group.

The transformation is done by moving one portion of data at a time in the source layout to the destination layout. Data is copied from the source volume to the temporary area, and data is removed from the source volume storage area in portions. The source volume storage area is then transformed to the new layout, and the data saved in the temporary area is written back to the new layout. This operation is repeated until all the storage and data in the source volume has been transformed to the new layout.

The default size of the temporary area used during the relayout depends on the size of the volume and the type of relayout. For volumes larger than 50MB, the

amount of temporary space that is required is usually 10% of the size of the volume, from a minimum of 50MB up to a maximum of 1GB. For volumes smaller than 50MB, the temporary space required is the same as the size of the volume.

The following error message displays the number of blocks required if there is insufficient free space available in the disk group for the temporary area:

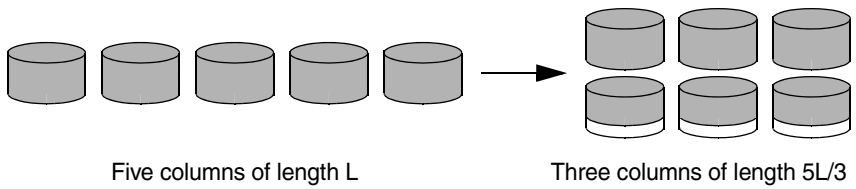
```
tmpsize too small to perform this relayout (nblks minimum
required)
```

You can override the default size used for the temporary area by using the `tmpsize` attribute to `vxassist`. See the `vxassist`(1M) manual page for more information.

As well as the temporary area, space is required for a temporary intermediate volume when increasing the column length of a striped volume. The amount of space required is the difference between the column lengths of the target and source volumes. For example, 20GB of temporary additional space is required to relayout a 150GB striped volume with 5 columns of length 30GB as 3 columns of length 50GB. In some cases, the amount of temporary space that is required is relatively large. For example, a relayout of a 150GB striped volume with 5 columns as a concatenated volume (with effectively one column) requires 120GB of space for the intermediate volume.

Additional permanent disk space may be required for the destination volumes, depending on the type of relayout that you are performing. This may happen, for example, if you change the number of columns in a striped volume. Figure 1-26 shows how decreasing the number of columns can require disks to be added to a volume. The size of the volume remains the same but an extra disk is needed to extend one of the columns.

Figure 1-26 Example of decreasing the number of columns in a volume

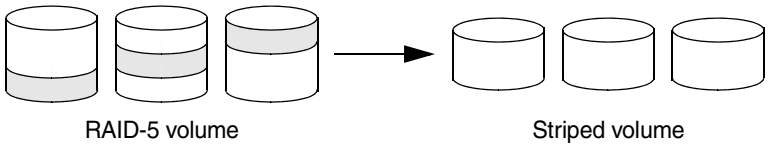


The following are examples of operations that you can perform using online relayout:

- Change a RAID-5 volume to a concatenated, striped, or layered volume (remove parity). See Figure 1-27 for an example. Note that removing parity

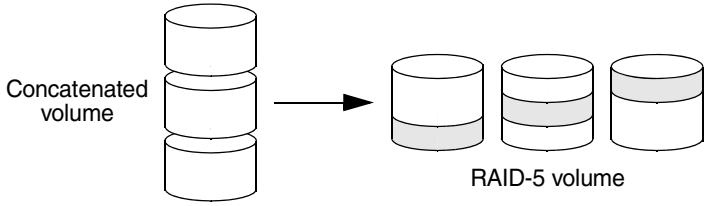
(shown by the shaded area) decreases the overall storage space that the volume requires.

Figure 1-27 Example of relayout of a RAID-5 volume to a striped volume



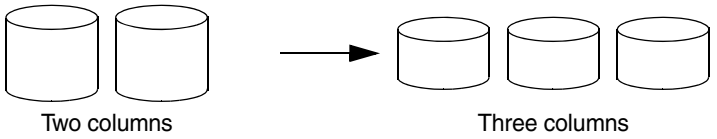
- Change a volume to a RAID-5 volume (add parity). See Figure 1-28 for an example. Note that adding parity (shown by the shaded area) increases the overall storage space that the volume requires.

Figure 1-28 Example of relayout of a concatenated volume to a RAID-5 volume



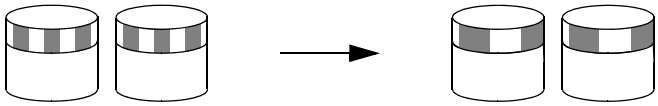
- Change the number of columns in a volume. See Figure 1-29 for an example. Note that the length of the columns is reduced to conserve the size of the volume.

Figure 1-29 Example of increasing the number of columns in a volume



- Change the column stripe width in a volume. See Figure 1-30 for an example.

Figure 1-30 Example of increasing the stripe width for the columns in a volume



For details of how to perform online relayout operations, see "Performing online relayout" on page 294. For information about the relayout transformations that are possible, see "Permitted relayout transformations" on page 295.

Limitations of online relayout

Note the following limitations of online relayout:

- Log plexes cannot be transformed.
- Volume snapshots cannot be taken when there is an online relayout operation running on the volume.
- Online relayout cannot create a non-layered mirrored volume in a single step. It always creates a layered mirrored volume even if you specify a non-layered mirrored layout, such as `mirror-stripe` or `mirror-concat`. Use the `vxassist convert` command to turn the layered mirrored volume that results from a relayout into a non-layered volume. See "Converting between layered and non-layered volumes" on page 300 for more information.
- Online relayout can be used only with volumes that have been created using the `vxassist` command or the Veritas Enterprise Administrator (VEA).
- The usual restrictions apply for the minimum number of physical disks that are required to create the destination layout. For example, mirrored volumes require at least as many disks as mirrors, striped and RAID-5 volumes require at least as many disks as columns, and striped-mirror volumes require at least as many disks as columns multiplied by mirrors.
- To be eligible for layout transformation, the plexes in a mirrored volume must have identical stripe widths and numbers of columns. Relayout is not possible unless you make the layouts of the individual plexes identical.
- Online relayout involving RAID-5 volumes is not supported for shareable disk groups in a cluster environment.
- Online relayout cannot transform sparse plexes, nor can it make any plex sparse. (A sparse plex is not the same size as the volume, or has regions that are not mapped to any subdisk.)

- The number of mirrors in a mirrored volume cannot be changed using relayout.
- Only one relayout may be applied to a volume at a time.

Transformation characteristics

Transformation of data from one layout to another involves rearrangement of data in the existing layout to the new layout. During the transformation, online relayout retains data redundancy by mirroring any temporary space used. Read and write access to data is not interrupted during the transformation.

Data is not corrupted if the system fails during a transformation. The transformation continues after the system is restored and both read and write access are maintained.

You can reverse the layout transformation process at any time, but the data may not be returned to the exact previous storage location. Any existing transformation in the volume must be stopped before doing a reversal.

You can determine the transformation direction by using the `vxrelayout status` *volume* command.

These transformations are protected against I/O failures if there is sufficient redundancy and space to move the data.

Transformations and volume length

Some layout transformations can cause the volume length to increase or decrease. If either of these conditions occurs, online relayout uses the `vxresize(1M)` command to shrink or grow a file system as described in "Resizing a volume" on page 284.

Volume resynchronization

When storing data redundantly and using mirrored or RAID-5 volumes, VxVM ensures that all copies of the data match exactly. However, under certain conditions (usually due to complete system failures), some redundant data on a volume can become inconsistent or *unsynchronized*. The mirrored data is not exactly the same as the original data. Except for normal configuration changes (such as detaching and reattaching a plex), this can only occur when a system crashes while data is being written to a volume.

Data is written to the mirrors of a volume in parallel, as is the data and parity in a RAID-5 volume. If a system crash occurs before all the individual writes complete, it is possible for some writes to complete while others do not. This can result in the data becoming unsynchronized. For mirrored volumes, it can cause two reads from the same region of the volume to return different results, if different mirrors are used to satisfy the read request. In the case of RAID-5 volumes, it can lead to parity corruption and incorrect data reconstruction.

VxVM needs to ensure that all mirrors contain exactly the same data and that the data and parity in RAID-5 volumes agree. This process is called *volume resynchronization*. For volumes that are part of the disk group that is automatically imported at boot time (usually aliased as the reserved system-wide disk group, `bootdg`), the resynchronization process takes place when the system reboots.

Not all volumes require resynchronization after a system failure. Volumes that were never written or that were quiescent (that is, had no active I/O) when the system failure occurred could not have had outstanding writes and do not require resynchronization.

Dirty flags

VxVM records when a volume is first written to and marks it as *dirty*. When a volume is closed by all processes or stopped cleanly by the administrator, and all writes have been completed, VxVM removes the dirty flag for the volume. Only volumes that are marked dirty when the system reboots require resynchronization.

Resynchronization process

The process of resynchronization depends on the type of volume. RAID-5 volumes that contain RAID-5 logs can “replay” those logs. If no logs are available, the volume is placed in reconstruct-recovery mode and all parity is regenerated. For mirrored volumes, resynchronization is done by placing the volume in recovery mode (also called *read-writeback recovery mode*).

Resynchronization of data in the volume is done in the background. This allows the volume to be available for use while recovery is taking place.

The process of resynchronization can impact system performance. The recovery process reduces some of this impact by spreading the recoveries to avoid stressing a specific disk or controller.

For large volumes or for a large number of volumes, the resynchronization process can take time. These effects can be addressed by using dirty region logging (DRL) and FastResync (fast mirror resynchronization) for mirrored volumes, or by ensuring that RAID-5 volumes have valid RAID-5 logs. See the sections "Dirty region logging" on page 60 and "FastResync" on page 66 for more information.

For raw volumes used by database applications, the SmartSync feature can be used if this is supported by the database vendor (see "SmartSync recovery accelerator" on page 62).

Dirty region logging

Note: If a version 20 DCO volume is associated with a volume, a portion of the DCO volume can be used to store the DRL log. There is no need to create a separate DRL log for a volume which has a version 20 DCO volume. For more information, see "DCO volume versioning" on page 68.

You need a full license to use this feature.

Dirty region logging (DRL), if enabled, speeds recovery of mirrored volumes after a system crash. DRL keeps track of the regions that have changed due to I/O writes to a mirrored volume. DRL uses this information to recover only those portions of the volume that need to be recovered.

If DRL is not used and a system failure occurs, all mirrors of the volumes must be restored to a consistent state. Restoration is done by copying the full contents of the volume between its mirrors. This process can be lengthy and I/O intensive. It may also be necessary to recover the areas of volumes that are already consistent.

Dirty region logs

DRL logically divides a volume into a set of consecutive regions, and maintains a log on disk where each region is represented by a status bit. This log records regions of a volume for which writes are pending. Before data is written to a region, DRL synchronously marks the corresponding status bit in the log as *dirty*. To enhance performance, the log bit remains set to dirty until the region

becomes the least recently accessed for writes. This allows writes to the same region to be written immediately to disk if the region's log bit is set to dirty.

On restarting a system after a crash, VxVM recovers only those regions of the volume that are marked as dirty in the dirty region log.

Log subdisks and plexes

DRL log subdisks store the dirty region log of a mirrored volume that has DRL enabled. A volume with DRL has at least one log subdisk; multiple log subdisks can be used to mirror the dirty region log. Each log subdisk is associated with one plex of the volume. Only one log subdisk can exist per plex. If the plex contains only a log subdisk and no data subdisks, that plex is referred to as a *log plex*.

The log subdisk can also be associated with a regular plex that contains data subdisks. In that case, the log subdisk risks becoming unavailable if the plex must be detached due to the failure of one of its data subdisks.

If the `vxassist` command is used to create a dirty region log, it creates a log plex containing a single log subdisk by default. A dirty region log can also be set up manually by creating a log subdisk and associating it with a plex. The plex then contains both a log and data subdisks.

Sequential DRL

Some volumes, such as those that are used for database replay logs, are written sequentially and do not benefit from delayed cleaning of the DRL bits. For these volumes, *sequential DRL* can be used to limit the number of dirty regions. This allows for faster recovery should a crash occur. However, if applied to volumes that are written to randomly, sequential DRL can be a performance bottleneck as it limits the number of parallel writes that can be carried out.

The maximum number of dirty regions allowed for sequential DRL is controlled by a tunable as detailed in the description of `voldrl_max_seq_dirty` in "Tunable parameters" on page 475.

Note: DRL adds a small I/O overhead for most write access patterns.

For details of how to configure DRL and sequential DRL, see "Adding traditional DRL logging to a mirrored volume" on page 281, and "Preparing a volume for DRL and instant snapshots" on page 275.

SmartSync recovery accelerator

The SmartSync feature of Veritas Volume Manager increases the availability of mirrored volumes by only resynchronizing changed data. (The process of resynchronizing mirrored databases is also sometimes referred to as *resilvering*.) SmartSync reduces the time required to restore consistency, freeing more I/O bandwidth for business-critical applications. If supported by the database vendor, the SmartSync feature uses an extended interface between VxVM volumes and the database software to avoid unnecessary work during mirror resynchronization. For example, Oracle® automatically takes advantage of SmartSync to perform database resynchronization when it is available.

Note: The SmartSync feature of Veritas Volume Manager is only applicable to databases that are configured on raw volumes. You cannot use it with volumes that contain file systems. Use an alternative solution such as the Oracle Resilvering feature of Veritas File System (VxFS).

You must configure volumes correctly to use SmartSync. For VxVM, there are two types of volumes used by the database, as follows:

- *Data volumes* are all other volumes used by the database (control files and tablespace files).
- *Redo log volumes* contain redo logs of the database.

SmartSync works with these two types of volumes differently, so they must be configured as described in the following sections.

To enable the use of SmartSync with database volumes in shared disk groups, set the value of the `volcvm_smartsync` tunable to `1`. For a description of `volcvm_smartsync`, see "Tunable parameters" on page 475.

Data volume configuration

The recovery takes place when the database software is started, not at system startup. This reduces the overall impact of recovery when the system reboots. Because the recovery is controlled by the database, the recovery time for the volume is the resilvering time for the database (that is, the time required to replay the redo logs).

Because the database keeps its own logs, it is not necessary for VxVM to do logging. Data volumes should be configured as mirrored volumes *without* dirty region logs. In addition to improving recovery time, this avoids any run-time I/O overhead due to DRL, and improves normal database write access.

Redo log volume configuration

A *redo log* is a log of changes to the database data. Because the database does not maintain changes to the redo logs, it cannot provide information about which sections require resilvering. Redo logs are also written sequentially, and since traditional dirty region logs are most useful with randomly-written data, they are of minimal use for reducing recovery time for redo logs. However, VxVM can reduce the number of dirty regions by modifying the behavior of its dirty region logging feature to take advantage of sequential access patterns. Sequential DRL decreases the amount of data needing recovery and reduces recovery time impact on the system.

The enhanced interfaces for redo logs allow the database software to inform VxVM when a volume is to be used as a redo log. This allows VxVM to modify the DRL behavior of the volume to take advantage of the access patterns. Since the improved recovery time depends on dirty region logs, redo log volumes should be configured as mirrored volumes *with* sequential DRL.

For additional information, see “Sequential DRL” on page 61.

Volume snapshots

Veritas Volume Manager provides the capability for taking an image of a volume at a given point in time. Such an image is referred to as a *volume snapshot*. Such snapshots should not be confused with file system snapshots, which are point-in-time images of a Veritas File System.

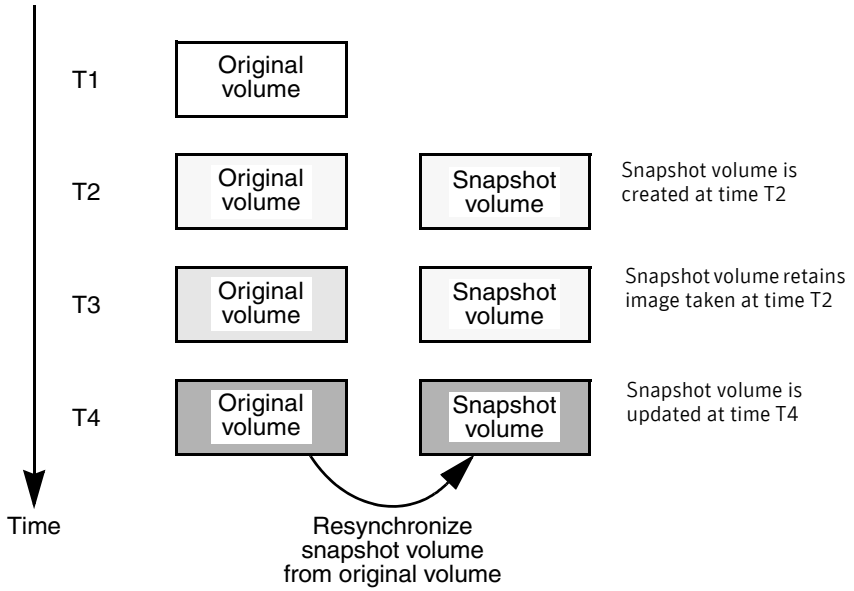
Figure 1-31 illustrates how a snapshot volume represents a copy of an original volume at a given point in time. Even though the contents of the original volume can change, the snapshot volume can be used to preserve the contents of the original volume as they existed at an earlier time.

The snapshot volume provides a stable and independent base for making backups of the contents of the original volume, or for other applications such as decision support. In the figure, the contents of the snapshot volume are eventually resynchronized with the original volume at a later point in time.

Another possibility is to use the snapshot volume to restore the contents of the original volume. This may be useful if the contents of the original volume have become corrupted in some way.

Note: If you choose to write to the snapshot volume, it may no longer be suitable for use in restoring the contents of the original volume.

Figure 1-31 Volume snapshot as a point-in-time image of a volume



The traditional type of volume snapshot in VxVM is of the *third-mirror break-off* type. This name comes from its implementation where a snapshot plex (or third mirror) is added to a mirrored volume. The contents of the snapshot plex are then synchronized from the original plexes of the volume. When this synchronization is complete, the snapshot plex can be detached as a snapshot volume for use in backup or decision support applications. At a later time, the snapshot plex can be reattached to the original volume, requiring a full resynchronization of the snapshot plex's contents. For more information about this type of snapshot, see "Traditional third-mirror break-off snapshots" on page 305.

The FastResync feature was introduced to track writes to the original volume. This tracking means that only a partial, and therefore much faster, resynchronization is required on reattaching the snapshot plex. In later releases, the snapshot model was enhanced to allow snapshot volumes to contain more than a single plex, reattachment of a subset of a snapshot volume's plexes, and persistence of FastResync across system reboots or cluster restarts.

For more information about FastResync, see "FastResync" on page 66.

Release 4.0 of VxVM introduced full-sized instant snapshots and space-optimized instant snapshots, which offer advantages over traditional third-

mirror snapshots such as immediate availability and easier configuration and administration. You can also use the third-mirror break-off usage model with full-sized snapshots, where this is necessary for write-intensive applications.

For more information, see the following sections:

- "Full-sized instant snapshots" on page 307.
- "Space-optimized instant snapshots" on page 309.
- "Emulation of third-mirror break-off snapshots" on page 310.
- "Linked break-off snapshot volumes" on page 311.

"Comparison of snapshot features" on page 65 compares the features that are supported by the different types of snapshot.

For more information about taking snapshots of a volume, see "Administering volume snapshots" on page 303, and the `vxsnap`(1M) and `vxassist`(1M) manual pages.

Comparison of snapshot features

The table, "Comparison of snapshot features for supported snapshot types" on page 65, compares the features of the various types of snapshots that are supported in VxVM.

Full-sized instant snapshots are easier to configure and offer more flexibility of use than do traditional third-mirror break-off snapshots. For preference, new volumes should be configured to use snapshots that have been created using the `vxsnap` command rather than using the `vxassist` command. Legacy volumes can also be reconfigured to use `vxsnap` snapshots, but this requires rewriting of administration scripts that assume the `vxassist` snapshot model.

If storage space is at a premium, space-optimized instant snapshots can be configured with some reduction of supported functionality. For example, space-optimized snapshots cannot be turned into independent volumes, nor can they be moved into a separate disk group for off-host processing.

Table 1-1 Comparison of snapshot features for supported snapshot types

Snapshot feature	Full-sized instant (vxsnap)	Space-optimized instant (vxsnap)	Break-off (vxassist or vxsnap)
Immediately available for use on creation	Yes	Yes	No
Requires less storage space than original volume	No	Yes	No

Table 1-1 Comparison of snapshot features for supported snapshot types

Snapshot feature	Full-sized instant (vxsnap)	Space-optimized instant (vxsnap)	Break-off (vxassist or vxsnap)
Can be reattached to original volume	Yes	No	Yes
Can be used to restore contents of original volume	Yes	Yes	Yes
Can quickly be refreshed without being reattached	Yes	Yes	No
Snapshot hierarchy can be split	Yes	No	No
Can be moved into separate disk group from original volume	Yes	No	Yes
Can be turned into an independent volume	Yes	No	Yes
FastResync ability persists across system reboots or cluster restarts	Yes	Yes	Yes
Synchronization can be controlled	Yes	No	No

FastResync

Note: You need a Veritas FlashSnap or FastResync license to use this feature.

The FastResync feature (previously called Fast Mirror Resynchronization or FMR) performs quick and efficient resynchronization of stale mirrors (a mirror that is not synchronized). This increases the efficiency of the VxVM snapshot mechanism, and improves the performance of operations such as backup and decision support applications. Typically, these operations require that the volume is quiescent, and that they are not impeded by updates to the volume by other activities on the system. To achieve these goals, the snapshot mechanism in VxVM creates an exact copy of a primary volume at an instant in time. After a

snapshot is taken, it can be accessed independently of the volume from which it was taken. In a clustered VxVM environment with shared access to storage, it is possible to eliminate the resource contention and performance overhead of using a snapshot simply by accessing it from a different node.

For details of how to enable FastResync on a per-volume basis, see "Enabling FastResync on a volume" on page 292.

FastResync enhancements

FastResync provides two fundamental enhancements to VxVM:

- FastResync optimizes mirror resynchronization by keeping track of updates to stored data that have been missed by a mirror. (A mirror may be unavailable because it has been *detached* from its volume, either automatically by VxVM as the result of an error, or directly by an administrator using a utility such as `vxplex` or `vxassist`. A *returning mirror* is a mirror that was previously detached and is in the process of being re-attached to its original volume as the result of the `vxrecover` or `vxplex att` operation.) When a mirror returns to service, only the updates that it has missed need to be re-applied to resynchronize it. This requires much less effort than the traditional method of copying all the stored data to the returning mirror.

 Once FastResync has been enabled on a volume, it does not alter how you administer mirrors. The only visible effect is that repair operations conclude more quickly.

- FastResync allows you to refresh and re-use snapshots rather than discard them. You can quickly re-associate (*snapback*) snapshot plexes with their original volumes. This reduces the system overhead required to perform cyclical operations such as backups that rely on the snapshot functionality of VxVM.

Non-persistent FastResync

Non-persistent FastResync allocates its change maps in memory. If non-persistent FastResync is enabled, a separate FastResync map is kept for the original volume and for each snapshot volume. Unlike a dirty region log (DRL), they do not reside on disk nor in persistent store. This has the advantage that updates to the FastResync map have little impact on I/O performance, as no disk updates needed to be performed. However, if a system is rebooted, the information in the map is lost, so a full resynchronization is required on snapback. This limitation can be overcome for volumes in cluster-shareable disk groups, provided that at least one of the nodes in the cluster remained running to preserve the FastResync map in its memory. However, a node crash in a High

Availability (HA) environment requires the full resynchronization of a mirror when it is reattached to its parent volume.

How non-persistent FastResync works with snapshots

The snapshot feature of VxVM takes advantage of FastResync change tracking to record updates to the original volume after a snapshot plex is created. After a snapshot is taken, the `snapback` option is used to reattach the snapshot plex. Provided that FastResync is enabled on a volume before the snapshot is taken, and that it is not disabled at any time before the snapshot is reattached, the changes that FastResync records are used to resynchronize the volume during the snapback. This considerably reduces the time needed to resynchronize the volume.

Non-Persistent FastResync uses a map in memory to implement change tracking. Each bit in the map represents a contiguous number of blocks in a volume's address space. The default size of the map is 4 blocks. The kernel tunable `vol_fmr_logsz` can be used to limit the maximum size in blocks of the map as described on "Tunable parameters" on page 475.

Persistent FastResync

Unlike non-persistent FastResync, persistent FastResync keeps the FastResync maps on disk so that they can survive system reboots, system crashes and cluster crashes. Persistent FastResync can also track the association between volumes and their snapshot volumes after they are moved into different disk groups. When the disk groups are rejoined, this allows the snapshot plexes to be quickly resynchronized. This ability is not supported by non-persistent FastResync. See "Reorganizing the contents of disk groups" on page 195 for details.

If persistent FastResync is enabled on a volume or on a snapshot volume, a *data change object* (DCO) and a *DCO volume* are associated with the volume.

DCO volume versioning

The internal layout of the DCO volume changed in VxVM 4.0 to support new features such as full-sized and space-optimized instant snapshots. Because the DCO volume layout is versioned, VxVM software continues to support the version 0 layout for legacy volumes. However, you must configure a volume to have a version 20 DCO volume if you want to take instant snapshots of the volume. Future releases of Veritas Volume Manager may introduce new versions of the DCO volume layout.

See "Determining the DCO version number" on page 277 for a description of how to find out the version number of a DCO that is associated with a volume.

Version 0 DCO volume layout

In VxVM releases 3.2 and 3.5, the DCO object only managed information about the FastResync maps. These maps track writes to the original volume and to each of up to 32 snapshot volumes since the last `snapshot` operation. Each plex of the DCO volume on disk holds 33 maps, each of which is 4 blocks in size by default.

Persistent FastResync uses the maps in a version 0 DCO volume on disk to implement change tracking. As for non-persistent FastResync, each bit in the map represents a *region* (a contiguous number of blocks) in a volume's address space. The size of each map can be changed by specifying the `dcolen` attribute to the `vxassist` command when the volume is created. The default value of `dcolen` is 132 1024-byte blocks (the plex contains 33 maps, each of length 4 blocks). To use a larger map size, multiply the desired map size by 33 to calculate the value of `dcolen` that you need to specify. For example, to use an 8-block map, you would specify `dcolen=264`. The maximum possible map size is 64 blocks, which corresponds to a `dcolen` value of 2112 blocks.

Note: The size of a DCO plex is rounded up to the nearest integer multiple of the disk group alignment value. The alignment value is 8KB for disk groups that support the Cross-platform Data Sharing (CDS) feature. Otherwise, the alignment value is 1 block.

Only traditional (third-mirror) volume snapshots that are administered using the `vxassist` command are supported for the version 0 DCO volume layout. Full-sized and space-optimized instant snapshots are not supported.

Version 20 DCO volume layout

In VxVM 4.0 and later releases, the DCO object is used not only to manage the FastResync maps, but also to manage DRL recovery maps (see "Dirty region logging" on page 60) and special maps called *copymaps* that allow instant snapshot operations to resume correctly following a system crash.

Each bit in a map represents a *region* (a contiguous number of blocks) in a volume's address space. A region represents the smallest portion of a volume for which changes are recorded in a map. A write to a single byte of storage anywhere within a region is treated in the same way as a write to the entire region.

The layout of a version 20 DCO volume includes an accumulator that stores the DRL map and a per-region state map for the volume, plus 32 per-volume maps (by default) including a DRL recovery map, and a map for tracking detaches that are initiated by the kernel due to I/O error. The remaining 30 per-volume maps

(by default) are used either for tracking writes to snapshots, or as copymaps. The size of the DCO volume is determined by the size of the regions that are tracked, and by the number of per-volume maps. Both the region size and the number of per-volume maps in a DCO volume may be configured when a volume is prepared for use with snapshots. The region size must be a power of 2 and be greater than or equal to 16KB.

As the accumulator is approximately 3 times the size of a per-volume map, the size of each plex in the DCO volume can be estimated from this formula:

```
DCO_plex_size = ( 3 + number_of_per-volume_maps ) * map_size
```

where the size of each map in bytes is:

```
map_size = 512 + ( volume_size / ( region_size * 8 ))
```

rounded up to the nearest multiple of 8KB. Note that each map includes a 512-byte header.

For the default number of 32 per-volume maps and region size of 64KB, a 10GB volume requires a map size of 24KB, and so each plex in the DCO volume requires 840KB of storage.

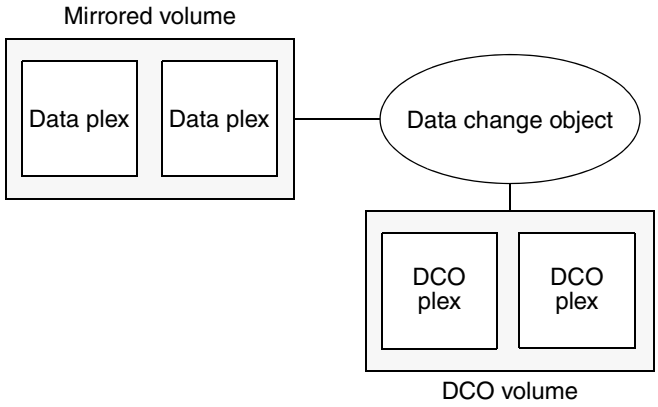
Note: Full-sized and space-optimized instant snapshots, which are administered using the `vxsnap` command, are supported for a version 20 DCO volume layout. The use of the `vxassist` command to administer traditional (third-mirror break-off) snapshots is not supported for a version 20 DCO volume layout.

How persistent FastResync works with snapshots

Persistent FastResync uses a map in a DCO volume on disk to implement change tracking. As for non-persistent FastResync, each bit in the map represents a contiguous number of blocks in a volume's address space.

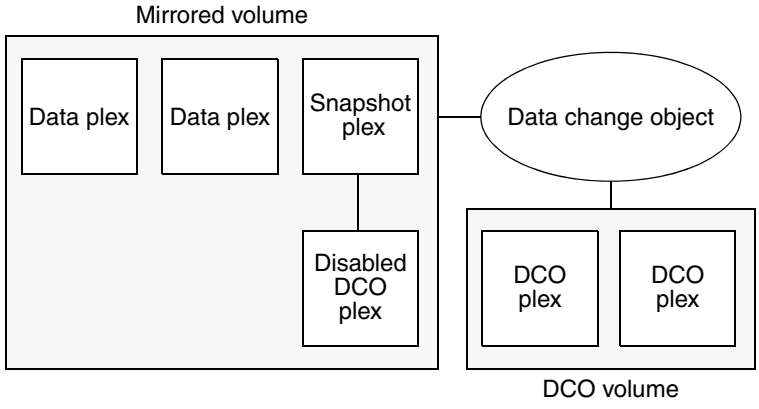
Figure 1-32 shows an example of a mirrored volume with two plexes on which Persistent FastResync is enabled. Associated with the volume are a DCO object and a DCO volume with two plexes.

Figure 1-32 Mirrored volume with persistent FastResync enabled



To create a traditional third-mirror snapshot or an instant (copy-on-write) snapshot, the `vxassist snapstart` or `vxsnap make` operation respectively is performed on the volume. This sets up a snapshot plex in the volume and associates a disabled DCO plex with it, as shown in Figure 1-33.

Figure 1-33 Mirrored volume after completion of a snapstart operation



Multiple snapshot plexes and associated DCO plexes may be created in the volume by re-running the `vxassist snapstart` command for traditional snapshots, or the `vxsnap make` command for space-optimized snapshots. You can create up to a total of 32 plexes (data and log) in a volume.

Note: Space-optimized instant snapshots do not require additional full-sized plexes to be created. Instead, they use a storage cache that typically requires only 10% of the storage that is required by full-sized snapshots. There is a trade-off in functionality in using space-optimized snapshots as described in "Comparison of snapshot features" on page 65. The storage cache is formed within a cache volume, and this volume is associated with a cache object. For convenience of operation, this cache can be shared by all the instant space-optimized snapshots within a disk group.

A traditional snapshot volume is created from a snapshot plex by running the `vxassist snapshot` operation on the volume. For instant snapshots, however, the `vxsnap make` command makes an instant snapshot volume immediately available for use. There is no need to run an additional command.

As illustrated in Figure 1-34, creation of the snapshot volume also sets up a DCO object and a DCO volume for the snapshot volume. This DCO volume contains the single DCO plex that was associated with the snapshot plex. If two snapshot plexes were taken to form the snapshot volume, the DCO volume would contain two plexes. For instant space-optimized snapshots, the DCO object and DCO volume are associated with a snapshot volume that is created on a cache object and not on a VM disk.

Associated with both the original volume and the snapshot volume are *snap objects*. The snap object for the original volume points to the snapshot volume, and the snap object for the snapshot volume points to the original volume. This allows VxVM to track the relationship between volumes and their snapshots even if they are moved into different disk groups.

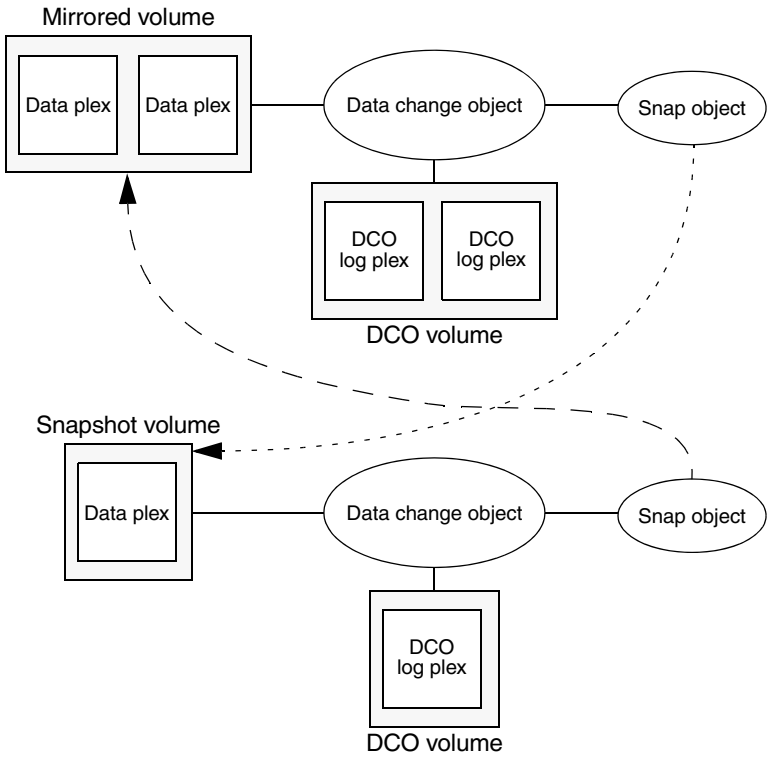
The snap objects in the original volume and snapshot volume are automatically deleted in the following circumstances:

- For traditional snapshots, the `vxassist snapback` operation is run to return all of the plexes of the snapshot volume to the original volume.
- For traditional snapshots, the `vxassist snapclear` operation is run on a volume to break the association between the original volume and the snapshot volume. If the volumes are in different disk groups, the command must be run separately on each volume.
- For full-sized instant snapshots, the `vxsnap reattach` operation is run to return all of the plexes of the snapshot volume to the original volume.
- For full-sized instant snapshots, the `vxsnap dis` or `vxsnap split` operations are run on a volume to break the association between the original volume and the snapshot volume. If the volumes are in different disk groups, the command must be run separately on each volume.

Note: The `vxsnap reattach`, `dis` and `split` operations are not supported for instant space-optimized snapshots.

See "Administering volume snapshots" on page 303, and the `vxsnap`(1M) and `vxassist`(1M) manual pages for more information.

Figure 1-34 Mirrored volume and snapshot volume after completion of a snapshot operation



Effect of growing a volume on the FastResync map

It is possible to grow the replica volume, or the original volume, and still use FastResync. According to the DCO volume layout, growing the volume has

different effects on the map that FastResync uses to track changes to the original volume:

- For a version 20 DCO volume, the size of the map is increased and the size of the region that is tracked by each bit in the map stays the same.
- For a version 0 DCO volume, the size of the map remains the same and the region size is increased.

In either case, the part of the map that corresponds to the grown area of the volume is marked as "dirty" so that this area is resynchronized. The `snapback` operation fails if it attempts to create an incomplete snapshot plex. In such cases, you must grow the replica volume, or the original volume, before invoking any of the commands `vxsnap reattach`, `vxsnap restore`, or `vxassist snapback`. Growing the two volumes separately can lead to a snapshot that shares physical disks with another mirror in the volume. To prevent this, grow the volume after the `snapback` command is complete.

FastResync limitations

The following limitations apply to FastResync:

- Persistent FastResync is supported for RAID-5 volumes, but this prevents the use of the relayout or resize operations on the volume while a DCO is associated with it.
- Neither non-persistent nor persistent FastResync can be used to resynchronize mirrors after a system crash. Dirty region logging (DRL), which can coexist with FastResync, should be used for this purpose. In VxVM 4.0 and later releases, DRL logs may be stored in a version 20 DCO volume.
- When a subdisk is relocated, the entire plex is marked "dirty" and a full resynchronization becomes necessary.
- If a snapshot volume is split off into another disk group, non-persistent FastResync cannot be used to resynchronize the snapshot plexes with the original volume when the disk group is rejoined with the original volume's disk group. Persistent FastResync must be used for this purpose.
- If you move or split an original volume (on which persistent FastResync is enabled) into another disk group, and then move or join it to a snapshot volume's disk group, you cannot use `vxassist snapback` to resynchronize traditional snapshot plexes with the original volume. This restriction arises because a snapshot volume references the original volume by its record ID at the time that the snapshot volume was created. Moving the original volume to a different disk group changes the volume's record ID, and so breaks the

association. However, in such a case, you can use the `vxplex snapback` command with the `-f` (force) option to perform the snapback.

Note: This restriction only applies to traditional snapshots. It does not apply to instant snapshots.

- Any operation that changes the layout of a replica volume can mark the FastResync change map for that snapshot "dirty" and require a full resynchronization during snapback. Operations that cause this include subdisk split, subdisk move, and online relayout of the replica. It is safe to perform these operations after the snapshot is completed. For more information, see the `vxvol` (1M), `vxassist` (1M), and `vxplex` (1M) manual pages.

Hot-relocation

Note: You need a full license to use this feature.

Hot-relocation is a feature that allows a system to react automatically to I/O failures on redundant objects (mirrored or RAID-5 volumes) in VxVM and restore redundancy and access to those objects. VxVM detects I/O failures on objects and relocates the affected subdisks. The subdisks are relocated to disks designated as *spare disks* and/or free space within the disk group. VxVM then reconstructs the objects that existed before the failure and makes them accessible again.

When a partial disk failure occurs (that is, a failure affecting only some subdisks on a disk), redundant data on the failed portion of the disk is relocated. Existing volumes on the unaffected portions of the disk remain accessible. For further details, see "Administering hot-relocation" on page 379.

Volume sets

Note: You need a full license to use this feature.

Volume sets are an enhancement to VxVM that allow several volumes to be represented by a single logical object. All I/O from and to the underlying volumes is directed via the I/O interfaces of the volume set. The volume set feature supports the multi-volume enhancement to Veritas File System (VxFS). This feature allows file systems to make best use of the different performance

and availability characteristics of the underlying volumes. For example, file system metadata could be stored on volumes with higher redundancy, and user data on volumes with better performance.

For more information about creating and administering volume sets, see “Creating and administering volume sets” on page 361.

Chapter 2

Administering disks

This chapter describes the operations for managing disks used by the Veritas Volume Manager (VxVM). This includes placing disks under VxVM control, initializing disks, mirroring the root disk, and removing and replacing disks.

Note: Most VxVM commands require superuser or equivalent privileges.

Disks that are controlled by the LVM subsystem cannot be used directly as VxVM disks, but they can be converted so that their volume groups and logical volumes become VxVM disk groups and volumes. For more information on conversion, see the *Veritas Volume Manager Migration Guide*.

For information about configuring and administering the dynamic multipathing (DMP) feature of VxVM that is used with multiported disk arrays, see "Administering dynamic multipathing (DMP)" on page 125.

Disk devices

When performing disk administration, it is important to understand the difference between a *disk name* and a *device name*.

When a disk is placed under VxVM control, a VM disk is assigned to it. You can define a symbolic *disk name* (also known as a *disk media name*) to refer to a VM disk for the purposes of administration. A disk name can be up to 31 characters long. If you do not assign a disk name, it defaults to `diskgroup##` where *diskgroup* is the name of the disk group to which the disk is being added, and *##* is a sequence number. Your system may use device names that differ from those given in the examples.

The *device name* (sometimes referred to as *devname* or *disk access name*) defines the name of a disk device as it is known to the operating system. In HP-UX 11i v3, the persistent (*agile*) forms of such devices are located in the `/dev/disk`

and /dev/rdisk directories. To maintain backward compatibility, HP-UX also creates legacy devices in the /dev/dsk and /dev/rdsk directories.

VxVM recreates disk devices for all paths in the operating system's hardware device tree as metadevices (*DMP nodes*) in the /dev/vx/dmp and /dev/vx/rdmp directories. The dynamic multipathing (DMP) feature of VxVM uses a DMP node to represent a disk that can be accessed by one or more physical paths, perhaps via different controllers. The number of access paths that are available depends on whether the disk is a single disk, or is part of a multiported disk array that is connected to a system. DMP nodes are not used by the native multipathing feature of HP-UX.

If a legacy device special file does not exist for the path to a LUN, DMP generates the DMP subpath name using the c#t#d# format, where the controller number in c# is set to 512 plus the instance number of the target path to which the LUN path belongs, the target is set to t0, and the device number in d# is set to the instance number of the LUN path. As the controller number is greater than 512, DMP subpath names that are generated in this way do not conflict with any legacy device names provided by the operating system. If a DMP subpath name has a controller number that is greater than 512, this implies that the operating system does not provide a legacy device special file for the device.

You can use the vxdisk utility to display the paths that are subsumed by a DMP metadevice, and to display the status of each path (for example, whether it is enabled or disabled).

For more information, see "Administering dynamic multipathing (DMP)" on page 125.

Device names may also be remapped as enclosure-based names as described in the following section.

Disk device naming in VxVM

There are two different methods of naming disk devices:

- Operating system-based naming
- Enclosure-based naming

Operating system-based naming

Under operating system-based naming, all disk devices except fabric mode disks are displayed either using the legacy c#t#d# format or the persistent disk## format. By default, VxVM commands display the names of these devices in the legacy format as these correspond to the names of the metanodes that are created by DMP.

The syntax of a legacy device name is `c#t#d#`, where `c#` represents a controller on a host bus adapter, `t#` is the target controller ID, and `d#` identifies a disk on the target controller.

Fabric mode disk devices are named as follows:

- Disks in supported disk arrays are named using the *enclosure name*`_#` format. For example, disks in the supported disk array name `FirstFloor` are named `FirstFloor_0`, `FirstFloor_1`, `FirstFloor_2` and so on. (You can use the `vxdmpadm` command to administer enclosure names.)
- Disks in the `DISKS` category (JBOD disks) are named using the `Disk_#` format.
- Disks in the `OTHER_DISKS` category (disks that are not multipathed by DMP) are named using the `fabric_#` format.

Enclosure-based naming

Enclosure-based naming operates as follows:

- All fabric or non-fabric disks in supported disk arrays are named using the *enclosure_name*`_#` format. For example, disks in the supported disk array, `enggdept` are named `enggdept_0`, `enggdept_1`, `enggdept_2` and so on. (You can use the `vxdmpadm` command to administer enclosure names. See "Administering DMP using vxdmpadm" on page 139 and the `vxdmpadm`(1M) manual page for more information.)
- Disks in the `DISKS` category (JBOD disks) are named using the `Disk_#` format.
- Disks in the `OTHER_DISKS` category (disks that are not multipathed by DMP) are named as follows:
 - Non-fabric disks are named using the `c#t#d#` or `disk##` format.
 - Fabric disks are named using the `fabric_#` format.

See "Changing the disk-naming scheme" on page 91 for details of how to switch between the operating system and enclosure based naming schemes.

To display the native OS device names of a VM disk (such as `mydg01`), use the following command:

```
# vxdisk path | egrep diskname
```

For information on how to rename an enclosure, see "Renaming an enclosure" on page 155.

For a description of disk categories, see "Disk categories" on page 83.

Private and public disk regions

Most VM disks have two regions:

private region A small area where configuration information is stored. A disk header label, configuration records for VxVM objects (such as volumes, plexes and subdisks), and an intent log for the configuration database are stored here. The default private region size is 32 megabytes (except for VxVM boot disk groups where the private region size must be 1 megabyte), which is large enough to record the details of several thousand VxVM objects in a disk group.

Under most circumstances, the default private region size should be sufficient. For administrative purposes, it is usually much simpler to create more disk groups that contain fewer volumes, or to split large disk groups into several smaller ones (as described in "Splitting disk groups" on page 205). If required, the value for the private region size may be overridden when you add or replace a disk using the `vxdiskadm` command.

Each disk that has a private region holds an entire copy of the configuration database for the disk group. The size of the configuration database for a disk group is limited by the size of the *smallest* copy of the configuration database on any of its member disks.

public region An area that covers the remainder of the disk, and which is used for the allocation of storage space to subdisks.

A disk's type identifies how VxVM accesses a disk, and how it manages the disk's private and public regions. The following disk access types are used by VxVM:

`simple` The public and private regions are on the same disk area (with the public area following the private area).

`nopriv` There is no private region (only a public region for allocating subdisks). This is the simplest disk type consisting only of space for allocating subdisks. Such disks are most useful for defining special devices (such as RAM disks, if supported) on which private region data would not persist between reboots. They can also be used to encapsulate disks where there is insufficient room for a private region. The disks cannot store configuration and log copies, and they do not support the use of the `vxdisk addregion` command to define reserved regions. VxVM cannot track the movement of `nopriv` disks on a SCSI chain or between controllers.

`auto` When the `vxconfigd` daemon is started, VxVM obtains a list of known disk device addresses from the operating system and configures disk access records for them automatically.

Auto-configured disks (with disk access type `auto`) support the following disk formats:

`cdsdisk` The disk is formatted as a Cross-platform Data Sharing (CDS) disk that is suitable for moving between different operating systems. This is the default format for disks that are not used to boot the system.Typically, most disks on a system are configured as this disk type. However, it is not a suitable format for boot, root or swap disks, for mirrors or hot-relocation spares of such disks, or for Extensible Firmware Interface (EFI) disks.

`hpdisk` The disk is formatted as a simple disk. This format can be applied to disks that are used to boot the system. The disk can be converted to a CDS disk if it was not initialized for use as a boot disk.

See the `vxcdsconvert`(1M) manual page for information about the utility that you can use to convert disks to the `cdsdisk` format.

Caution: The CDS disk format is incompatible with EFI disks. If a disk is initialized by VxVM as a CDS disk, the CDS header occupies the portion of the disk where the partition table would usually be located. If you subsequently use a command such as `fdisk` to create a partition table on a CDS disk, this erases the CDS information and could cause data corruption.

By default, auto-configured non-EFI disks are formatted as `cdsdisk` disks when they are initialized for use with VxVM. You can change the default format by using the `vxdiskadm`(1M) command to update the `/etc/default/vxdisk` defaults file as described in "Displaying and changing default disk layout attributes" on page 97. See the `vxdisk`(1M) manual page for details of the usage of this file, and for more information about disk types and their configuration.

Auto-configured EFI disks are formatted as `hpdisk` disks by default.

Discovering and configuring newly added disk devices

When you physically connect new disks to a host or when you zone new fibre channel devices to a host, you can use the `vxdctl enable` command to rebuild the volume device node directories and to update the DMP internal database to reflect the new state of the system.

To reconfigure the DMP database, first run `ioscan` followed by `insf` to make the operating system recognize the new disks, and then invoke the `vxdctl enable` command. See the `vxdctl`(1M) manual page for more information.

You can also use the `vxdisk scandisks` command to scan devices in the operating system device tree, and to initiate dynamic reconfiguration of multipathed disks.

If you want VxVM to scan only for new devices that have been added to the system, and not for devices that have been enabled or disabled, specify the `-f` option to either of the commands, as shown here:

```
# vxdctl -f enable
# vxdisk -f scandisks
```

However, a complete scan is initiated if the system configuration has been modified by changes to:

- Installed array support libraries.
- The devices that are listed as being excluded from use by VxVM.
- DISKS (JBOD), SCSI3, or foreign device definitions.

See the `vxdctl`(1M) and `vxdisk`(1M) manual pages for more information.

Partial device discovery

The Dynamic Multipathing (DMP) feature of VxVM supports partial device discovery where you can include or exclude sets of disks or disks attached to controllers from the discovery process.

The `vxdisk scandisks` command rescans the devices in the OS device tree and triggers a DMP reconfiguration. You can specify parameters to `vxdisk scandisks` to implement partial device discovery. For example, this command makes VxVM discover newly added devices that were unknown to it earlier:

```
# vxdisk scandisks new
```

The next example discovers fabric devices:

```
# vxdisk scandisks fabric
```

The following command scans for the devices `c1t1d0` and `c2t2d0`:

```
# vxdisk scandisks device=c1t1d0,c2t2d0
```

Alternatively, you can specify a ! prefix character to indicate that you want to scan for all devices *except* those that are listed:

```
# vxdisk scandisks !device=c1t1d0,c2t2d0
```

You can also scan for devices that are connected (or not connected) to a list of logical or physical controllers. For example, this command discovers and configures all devices except those that are connected to the specified logical controllers:

```
# vxdisk scandisks !ctlr=c1,c2
```

The next command discovers devices that are connected to the specified physical controller:

```
# vxdisk scandisks pctlr=8/12.8.0.255.0
```

Note: The items in a list of physical controllers are separated by + characters.

You can use the command `vxdmpadm getctlr all` to obtain a list of physical controllers.

You can specify only one selection argument to the `vxdisk scandisks` command. Specifying multiple options results in an error.

For more information, see the `vxdisk`(1M) manual page.

Discovering disks and dynamically adding disk arrays

You can dynamically add support for a new type of disk array which has been developed by a third-party vendor. The support comes in the form of vendor-supplied libraries, and is added to an HP-UX system by using the `swinstall` command.

Disk categories

Disk arrays that have been certified for use with Veritas Volume Manager are supported by an array support library (ASL), and are categorized by the vendor ID string that is returned by the disks (for example, "`HITACHI`").

Disks in JBODs for which DMP (see "Administering dynamic multipathing (DMP)" on page 125) can be supported in Active/Active mode, and which are capable of being multipathed, are placed in the `DISKS` category. Disks in unsupported arrays can be placed in this category by following the steps given in "Adding unsupported disk arrays to the DISKS category" on page 87.

Disks in JBODs that do not fall into any supported category, and which are not capable of being multipathed by DMP are placed in the `OTHER_DISKS` category.

Adding support for a new disk array

The following example illustrates how to add support for a new disk array named `vrtsda` to an HP-UX system using a vendor-supplied package on a mounted CD-ROM:

```
# swinstall -s /cdrom vrtsda
```

The new disk array does not need to be already connected to the system when the package is installed. If any of the disks in the new disk array are subsequently connected, first run the `ioscan` command, and then run either the `vxdisk scandisks` or the `vxdctl enable` command to include the devices in the VxVM device list.

Enabling discovery of new devices

To have VxVM discover a new disk array, use the following command:

```
# vxdctl enable
```

This command scans all of the disk devices and their attributes, updates the VxVM device list, and reconfigures DMP with the new device database. There is no need to reboot the host.

Note: This command ensures that dynamic multipathing is set up correctly on the array. Otherwise, VxVM treats the independent paths to the disks as separate devices, which can result in data corruption.

Removing support for a disk array

To remove support for the `vrtsda` disk array, use the following command:

```
# swremove vrtsda
```

If the arrays remain physically connected to the host after support has been removed, they are listed in the `OTHER_DISKS` category, and the volumes remain available.

Third-party driver coexistence

The third-party driver (TPD) coexistence feature of VxVM allows I/O that is controlled by third-party multipathing drivers to bypass DMP while retaining the monitoring capabilities of DMP. Provided that a suitable ASL is available, devices that use TPDs can be discovered without requiring you to set up a specification file, or to run a special command. In previous releases, VxVM only supported TPD coexistence if the code of the third-party driver was intrusively modified. The new TPD coexistence feature maintains backward compatibility with such methods, but it also permits coexistence without require any change in a third-party multipathing driver.

See “Changing device naming for TPD-controlled enclosures” on page 94 for information on how to change the form of TPD device names that are displayed by VxVM.

See “Displaying information about TPD-controlled devices” on page 143 for details of how to find out the TPD configuration information that is known to DMP.

Administering the Device Discovery Layer

Dynamic addition of disk arrays is possible because of the existence of the Device Discovery Layer (DDL) which is a facility for discovering disks and their attributes that are required for VxVM and DMP operations. The DDL is administered using the `vxddladm` utility, which can be used to perform the following tasks:

- List the types of arrays that are supported.
- Add support for an array to DDL.
- Remove support for an array from DDL.
- List information about excluded disk arrays.
- List disks that are supported in the `DISKS` (JBOD) category.
- Add disks from different vendors to the `DISKS` category.
- Remove disks from the `DISKS` category.
- Add disks as foreign devices.

The following sections explain these tasks in more detail. For further information, see the `vxddladm`(1M) manual page.

Listing details of supported disk arrays

To list all currently supported disk arrays, use the following command:

```
# vxddladm listsupport all
```

Note: Use this command to obtain values for the `vid` and `pid` attributes that are used with other forms of the `vxddladm` command.

To display more detailed information about a particular array library, use this form of the command:

```
# vxddladm listsupport libname=library_name.sl
```

This command displays the vendor ID (VID), product IDs (PIDs) for the arrays, array types (for example, A/A or A/P), and array names. The following is sample output.

```
# vxddladm listsupport libname=libvxfujitsu.so
ATTR_NAME            ATTR_VALUE
=================================================
LIBNAME              libvxfujitsu.so
VID                  vendor
PID                  GR710, GR720, GR730
                     GR740, GR820, GR840
ARRAY_TYPE           A/A, A/P
ARRAY_NAME           FJ_GR710, FJ_GR720, FJ_GR730
                     FJ_GR740, FJ_GR820, FJ_GR840
```

Excluding support for a disk array library

To exclude all arrays that depend on a particular array library from participating in device discovery, use the following command:

```
# vxddladm excludearray libname=libvxenc.sl
```

This example excludes support for disk arrays that depends on the library `libvxenc.sl`. You can also exclude support for disk arrays from a particular vendor, as shown in this example:

```
# vxddladm excludearray vid=ACME pid=X1
```

For more information about excluding disk array support, see the `vxddladm` (1M) manual page.

Re-including support for an excluded disk array library

If you have excluded support for all arrays that depend on a particular disk array library, you can use the `includearray` keyword to remove the entry from the exclude list, as shown in the following example:

```
# vxddladm includearray libname=libvxenc.sl
```

This command adds the array library to the database so that the library can once again be used in device discovery. If `vxconfigd` is running, you can use the `vxdisk scandisks` command to discover the arrays and add their details to the database.

Listing excluded disk arrays

To list all disk arrays that are currently excluded from use by VxVM, use the following command:

```
# vxddladm listexclude
```

Listing supported disks in the DISKS category

To list disks that are supported in the DISKS (JBOD) category, use the following command:

```
# vxddladm listjbod
```

Adding unsupported disk arrays to the DISKS category

Disk arrays should be added as JBOD devices if no ASL is available for the array.

JBODs are assumed to be Active/Active (A/A) unless otherwise specified. If a suitable ASL is not available, an A/A-A, A/P or A/PF array must be claimed as an Active/Passive (A/P) JBOD to prevent path delays and I/O failures.

Caution: The procedure in this section ensures that Dynamic Multipathing (DMP) is set up correctly on an array that is not supported by Veritas Volume Manager. If the native multipathing in HP-UX 11i v3 does not recognize such an array as being multipathed, Veritas Volume Manager treats the independent legacy paths to the disks as separate devices, which can result in data corruption.

To add an unsupported disk array

1. Use the following command to identify the vendor ID and product ID of the disks in the array:

   ```
   # /etc/vx/diag.d/vxdmpinq device_name
   ```

 where *device_name* is the device name of one of the disks in the array (for example, /dev/rdsk/c1t20d0). Note the values of the vendor ID (VID) and product ID (PID) in the output from this command. For Fujitsu disks, also note the number of characters in the serial number that is displayed. The following is sample output:

   ```
   # /etc/vx/diag.d/vxdmpinq /dev/rdsk/c1t20d0
   Vendor id (VID)      : SEAGATE
   Product id (PID):      ST318404LSUN18G
   Revision             : 8507
   Serial Number        : 0025T0LA3H
   ```

 In this example, the vendor ID is SEAGATE and the product ID is ST318404LSUN18G.

2. Stop all applications, such as databases, from accessing VxVM volumes that are configured on the array, and unmount all VxFS file systems and checkpoints that are configured on the array.

3. If the array is of type A/A-A, A/P or A/PF, configure it in autotrespass mode.

4. Enter the following command to add a new JBOD category:

   ```
   # vxddladm addjbod vid=vendorid pid=productid \
   ```

[length=*serialno_length*] [policy=ap]

where *vendorid* and *productid* are the VID and PID values that you found from the previous step. For example, *vendorid* might be `FUJITSU`, `IBM`, or `SEAGATE`. For Fujitsu devices, you must also specify the number of characters in the serial number as the argument to the `length` argument (for example, 10). If the array is of type A/A-A, A/P or A/PF, you must also specify the `policy=ap` attribute.

Continuing the previous example, the command to define an array of disks of this type as a JBOD would be:

```
# vxddladm addjbod vid=SEAGATE pid=ST318404LSUN18G
```

5 Use the `vxdctl enable` command to bring the array under VxVM control.

```
# vxdctl enable
```

See "Enabling discovery of new devices" on page 84.

6 To verify that the array is now supported, enter the following command:

```
# vxddladm listjbod
```

The following is sample output from this command for the example array:

```
VID       PID       Opcode  Page Code   Page Offset   SNO length
==============================================================
SEAGATE   ALL PIDs  18      -1           36          12
```

7 To verify that the array is recognized, use the `vxdmpadm listenclosure` command as shown in the following sample output for the example array:

```
# vxdmpadm listenclosure all
ENCLR_NAME      ENCLR_TYPE      ENCLR_SNO       STATUS
============================================================
OTHER_DISKS     OTHER_DISKS     OTHER_DISKS     CONNECTED
Disk            Disk            DISKS           CONNECTED
```

The enclosure name and type for the array are both shown as being set to `Disk`. You can use the `vxdisk list` command to display the disks in the array:

```
# vxdisk list
DEVICE    TYPE          DISK        GROUP         STATUS
Disk_0    auto:none     -           -             online invalid
Disk_1    auto:none     -           -             online invalid
...
```

8 To verify that the DMP paths are recognized, use the `vxdmpadm getdmpnode` command as shown in the following sample output for the example array:

```
# vxdmpadm getdmpnode enclosure=Disk
NAME      STATE     ENCLR-TYPE  PATHS   ENBL    DSBL  ENCLR-NAME
============================================================
Disk_0    ENABLED   Disk        2       2         0   Disk
Disk_1    ENABLED   Disk        2       2         0   Disk
...
```

This shows that there are two paths to the disks in the array.

For more information, enter the command `vxddladm help addjbod`, or see the `vxddladm`(1M) and `vxdmpadm`(1M) manual pages.

Removing disks from the DISKS category

To remove disks from the `DISKS` (JBOD) category, use the `vxddladm` command with the `rmjbod` keyword. The following example illustrates the command for removing disks supplied by the vendor, Seagate:

```
# vxddladm rmjbod vid=SEAGATE
```

Adding foreign devices

DDL may not be able to discover some devices that are controlled by third-party drivers, such as those that provide multipathing or RAM disk capabilities. For these devices it may be preferable to use the multipathing capability that is provided by the third-party drivers for some arrays rather than using the Dynamic Multipathing (DMP) feature. Such *foreign devices* can be made available as simple disks to VxVM by using the `vxddladm addforeign` command. This also has the effect of bypassing DMP for handling I/O. The following example shows how to add entries for block and character devices in the specified directories:

```
# vxddladm addforeign blockdir=/dev/foo/dsk \
  chardir=/dev/foo/rdsk
```

By default, this command suppresses any entries for matching devices in the OS-maintained device tree that are found by the autodiscovery mechanism. You can override this behavior by using the `-f` and `-n` options as described on the `vxddladm`(1M) manual page.

After adding entries for the foreign devices, use either the `vxdisk scandisks` or the `vxdctl enable` command to discover the devices as simple disks. These disks then behave in the same way as autoconfigured disks.

The foreign device mechanism was introduced in VxVM 4.0 to support non-standard devices such as RAM disks, some solid state disks, and pseudo-devices such as EMC PowerPath. This mechanism has a number of limitations:

- A foreign device is always considered as a disk with a single path. Unlike an autodiscovered disk, it does not have a DMP node.
- It is not supported for shared disk groups in a clustered environment. Only standalone host systems are supported.
- It is not supported for Persistent Group Reservation (PGR) operations.
- It is not under the control of DMP, so enabling of a failed disk cannot be automatic, and DMP administrative commands are not applicable.

- Enclosure information is not available to VxVM. This can reduce the availability of any disk groups that are created using such devices.
- EFI disks that are under the control of HP-UX native multipathing cannot be initialized as foreign disks. You must migrate the system to DMP, initialize the disk as an EFI disk, and then migrate the system back to HP-UX native multipathing.
 See "Migrating between DMP and HP-UX native multipathing" on page 130.
- Foreign devices, such as HP-UX native multipathing metanodes, do not have enclosures, controllers or DMP nodes that can be administered using VxVM commands. An error message is displayed if you attempt to use the `vxddladm` or `vxdmpadm` commands to administer such devices while HP-UX native multipathing is configured.
- The I/O Fencing and Cluster File System features are not supported for foreign devices.

If a suitable ASL is available for an array, these limitations are removed.

See "Third-party driver coexistence" on page 84.

Placing disks under VxVM control

When you add a disk to a system that is running VxVM, you need to put the disk under VxVM control so that VxVM can control the space allocation on the disk. Unless you specify a disk group, VxVM places new disks in a default disk group according to the rules given in "Rules for determining the default disk group" on page 168.

The method by which you place a disk under VxVM control depends on the circumstances:

- If the disk is new, it must be *initialized* and placed under VxVM control. You can use the menu-based `vxdiskadm` utility to do this.

 Caution: Initialization destroys existing data on disks.

- If the disk is not needed immediately, it can be initialized (but not added to a disk group) and reserved for future use. To do this, enter **`none`** when asked to name a disk group. Do not confuse this type of "spare disk" with a hot-relocation spare disk.
- If the disk was previously initialized for future use by VxVM, it can be reinitialized and placed under VxVM control.
- If the disk was previously used for a file system, VxVM prompts you to confirm that you really want to destroy the file system.

- If the disk was previously in use by the LVM subsystem, you can preserve existing data while still letting VxVM take control of the disk. This is accomplished using *conversion*. With conversion, the virtual layout of the data is fully converted to VxVM control (see the *Veritas Volume Manager Migration Guide*).
- If the disk was previously in use by the LVM subsystem, but you do not want to preserve the data on it, use the LVM command, `pvremove`, before attempting to initialize the disk for VxVM.
- Multiple disks on one or more controllers can be placed under VxVM control simultaneously. Depending on the circumstances, all of the disks may not be processed the same way.

It is possible to configure the `vxdiskadm` utility not to list certain disks or controllers as being available. For example, this may be useful in a SAN environment where disk enclosures are visible to a number of separate systems.

To exclude a device from the view of VxVM, select item 17 (`Prevent multipathing/Suppress devices from VxVM's view`) from the `vxdiskadm` main menu. See "Disabling and enabling multipathing for specific devices" on page 133 for details.

Changing the disk-naming scheme

You can either use enclosure-based naming for disks or the operating system's naming convention. Select menu item 20 from the `vxdiskadm` main menu to change the disk-naming scheme that you want VxVM to use. When prompted, enter **`y`** to change the naming scheme. For operating system based naming, you are asked to select between default, legacy or new device names. The `vxconfigd` daemon is then restarted to bring the new disk naming scheme into effect.

VxVM commands display device names according the current naming scheme. If operating system-based naming is selected, all VxVM commands that list DMP node devices will display device names according to the mode that is specified.

Mode	Format of output from VxVM command
`default`	The same format is used as in the input to the command (if this can be determined). Otherwise, legacy names are used. This is the default mode.
`legacy`	Only legacy names are displayed.
`new`	Only new (agile) names are displayed.

Alternatively, you can change the naming scheme from the command line. The following commands select enclosure-based and operating system-based naming respectively:

```
# vxddladm set namingscheme=ebn [persistence={yes|no}]
# vxddladm set namingscheme=osn [mode={default|legacy|new}] \
  [persistence={yes|no}]
```

The optional `persistence` argument allows you to select whether the DDL-assigned names of disk devices that are displayed by VxVM remain unchanged after disk hardware has been reconfigured and the system rebooted. By default, both enclosure-based naming and operating system-based naming are persistent.

The following examples illustrate the use of `vxddladm set namingscheme` command:

```
# vxddladm set namingscheme=osn mode=default
# vxdmpadm getlungroup dmpnodename=c1t65d0
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
c1t65d0   ENABLED   Disk        2        2       0      Disk

# vxdmpadm getlungroup dmpnodename=disk25
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
disk25    ENABLED   Disk        2        2       0      Disk

# vxddladm set namingscheme=osn mode=legacy
# vxdmpadm getlungroup dmpnodename=c1t65d0
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
c1t65d0   ENABLED   Disk        2        2       0      Disk

# vxdmpadm getlungroup dmpnodename=disk25
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
c1t65d0   ENABLED   Disk        2        2       0      Disk

# vxddladm set namingscheme=osn mode=new
# vxdmpadm getlungroup dmpnodename=c1t65d0
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
disk25    ENABLED   Disk        2        2       0      Disk

# vxdmpadm getlungroup dmpnodename=disk25
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
disk25    ENABLED   Disk        2        2       0      Disk

# vxddladm set namingscheme=ebn
# vxdmpadm getlungroup dmpnodename=c1t65d0
VxVM vxdmpadm ERROR V-5-1-10910  Invalid da-name
```

```
# vxdmpadm getlungroup dmpnodename=disk25
VxVM vxdmpadm ERROR V-5-1-10910  Invalid da-name

# vxdmpadm getlungroup dmpnodename=Disk_11
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
Disk_11   ENABLED   Disk        2        2       0      Disk
```

To find out which sort of naming is currently enabled, use the `vxddladm get namingscheme` command, as shown in the following example:

```
# vxddladm get namingscheme
NAMING_SCHEME       PERSISTENCE         MODE
=============================================
OS Native           Yes                 New
```

The effect of enabling persistent device names in conjunction with operating system-based naming is discussed in "Regenerating persistent device names" on page 93.

Regenerating persistent device names

The persistent device naming feature makes the DDL-assigned names of disk devices persistent across system reboots. If operating system-based naming is selected, each disk name is usually set to the name of one of the paths to the disk. After hardware reconfiguration and a subsequent reboot, the operating system may generate different names for the paths to the disks. As DDL assigns persistent disk names using the persistent device name database that was generated during a previous boot session, the disk names may no longer correspond to the actual paths. This does not prevent the disks from being used, but the association between the disk name and one of its paths is lost.

To find the relationship between a disk and its paths, run one of the following commands:

```
# vxdmpadm getsubpaths dmpnodename=disk_access_name
# vxdisk list disk_access_name
```

To update the disk names so that they correspond to the new path names

1. Remove the file that contains the existing persistent device name database:

   ```
   # rm /etc/vx/disk.info
   ```

2. Restart the VxVM configuration demon:

   ```
   # vxconfigd -k
   ```

 This regenerates the persistent name database.

Changing device naming for TPD-controlled enclosures

Note: This feature is available only if the default disk-naming scheme is set to use operating system-based naming, and the TPD-controlled enclosure does not contain fabric disks.

For disk enclosures that are controlled by third-party drivers (TPD) whose coexistence is supported by an appropriate ASL, the default behavior is to assign device names that are based on the TPD-assigned node names. You can use the `vxdmpadm` command to switch between these names and the device names that are known to the operating system:

```
# vxdmpadm setattr enclosure enclosure tpdmode=native|pseudo
```

The argument to the `tpdmode` attribute selects names that are based on those used by the operating system (`native`), or TPD-assigned node names (`pseudo`).

If `tpdmode` is set to `native`, the path with the smallest device number is displayed.

Discovering the association between enclosure and OS based disk names

If you enable enclosure-based naming, and use the `vxprint` command to display the structure of a volume, it shows enclosure-based disk device names (disk access names) rather than operating system-based names. To discover the operating system-based names that are associated with a given enclosure-based disk name, use either of the following commands:

```
# vxdisk list enclosure-based_name
# vxdmpadm getsubpaths dmpnodename=enclosure-based_name
```

For example, to find the physical device that is associated with disk `ENC0_21`, the appropriate commands would be:

```
# vxdisk list ENC0_21
# vxdmpadm getsubpaths dmpnodename=ENC0_21
```

To obtain the full pathname for the block and character disk device from these commands, append the displayed device name to `/dev/vx/dmp` or `/dev/vx/rdmp`.

Issues regarding persistent simple or nopriv disks with enclosure-based naming

If you change from *c#t#d#* based naming to enclosure-based naming, persistent simple or nopriv disks may be put in the "`error`" state and cause VxVM objects on those disks to fail. If this happens, use the following procedures to correct the problem:

- Persistent simple or nopriv disks in the boot disk group
- Persistent simple or nopriv disks in non-boot disk groups

These procedures use the `vxdarestore` utility to handle errors in persistent simple and nopriv disks that arise from changing to the enclosure-based naming scheme. You do not need to perform either procedure if the devices on which any simple or nopriv disks are present are not automatically configured by VxVM (for example, non-standard disk devices such as ramdisks).

Note: The disk access records for simple disks are either persistent or non-persistent. The disk access record for a persistent simple disk is stored in the disk's private region. The disk access record for a non-persistent simple disk is automatically configured in memory at VxVM startup. A simple disk has a non-persistent disk access record if `autoconfig` is included in the `flags` field that is displayed by the `vxdisk list` *disk_access_name* command. If the `autoconfig` flag is not present, the disk access record is persistent. Nopriv disks are always persistent.

Note: You cannot run `vxdarestore` if c#t#d# naming is in use. Additionally, `vxdarestore` does not handle failures on persistent simple/nopriv disks that are caused by renaming enclosures, by hardware reconfiguration that changes device names. or by removing support from the JBOD category for disks that belong to a particular vendor when enclosure-based naming is in use.

For more information about the `vxdarestore` command, see the `vxdarestore`(1M) manual page.

Persistent simple or nopriv disks in the boot disk group

If *all* persistent simple and nopriv disks in the boot disk group (usually aliased as `bootdg`) go into the error state, the `vxconfigd` daemon is disabled after the naming scheme change.

To remove the error state for persistent simple or nopriv disks in the boot disk group

1. Use vxdiskadm to change back to c#t#d# naming.
2. Enter the following command to restart the VxVM configuration daemon:

```
# vxconfigd -kr reset
```

3 If you want to use enclosure-based naming, use `vxdiskadm` to add a non-persistent simple disk to the `bootdg` disk group, change back to the enclosure-based naming scheme, and then run the following command:

```
# /etc/vx/bin/vxdarestore
```

Note: If not all the disks in `bootdg` go into the error state, you need only run `vxdarestore` to restore the disks that are in the error state and the objects that they contain.

Persistent simple or nopriv disks in non-boot disk groups

If an imported disk group, other than `bootdg`, consists *only* of persistent simple and/or nopriv disks, it is put in the "`online dgdisabled`" state after the change to the enclosure-based naming scheme.

To remove the error state for persistent simple or nopriv disks in non-boot disk groups

1 Deport the disk group using the following command:

```
# vxdg deport diskgroup
```

2 Use the `vxdarestore` command to restore the failed disks, and to recover the objects on those disks:

```
# /etc/vx/bin/vxdarestore
```

3 Re-import the disk group using the following command:

```
# vxdg import diskgroup
```

Installing and formatting disks

Depending on the hardware capabilities of your disks and of your system, you may either need to shut down and power off your system before installing the disks, or you may be able to hot-insert the disks into the live system. Many operating systems can detect the presence of the new disks on being rebooted. If the disks are inserted while the system is live, you may need to enter an operating system-specific command to notify the system.

If the disks require low or intermediate-level formatting before use, use the operating system-specific formatting command to do this.

Note: SCSI disks are usually preformatted. Reformatting is needed only if the existing formatting has become damaged.

The following sections provide detailed examples of how to use the `vxdiskadm` utility to place disks under VxVM control in various ways and circumstances.

Displaying and changing default disk layout attributes

To display or change the default values for initializing disks, select menu item `21 (Change/display the default disk layout)` from the `vxdiskadm` main menu. For disk initialization, you can change the default format and the default length of the private region.

The attribute settings for initializing disks are stored in the file, `/etc/default/vxdisk`.

See the `vxdisk`(1M) manual page for more information.

Adding a disk to VxVM

Formatted disks being placed under VxVM control may be new or previously used outside VxVM. The set of disks can consist of all disks on the system, all disks on a controller, selected disks, or a combination of these.

Depending on the circumstances, all of the disks may not be processed in the same way.

Caution: Initialization does not preserve data on disks.

When initializing multiple disks at one time, it is possible to exclude certain disks or certain controllers. To exclude disks, list the names of the disks to be excluded in the file `/etc/vx/disks.exclude` before the initialization. You can exclude all disks on specific controllers from initialization by listing those controllers in the file `/etc/vx/cntrls.exclude`.

To initialize disks for VxVM use

1. Select menu item `1 (Add or initialize one or more disks)` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk device name of the disk to be added to VxVM control (or enter **`list`** for a list of disks):

```
Add or initialize disks
Menu: VolumeManager/Disk/AddDisks

Use this operation to add one or more disks to a disk group.
You can add the selected disks to an existing disk group or to
a new disk group that will be created as a part of the
operation. The selected disks may also be added to a disk group
as spares. Or they may be added as nohotuses to be excluded
from hot-relocation use. The selected disks may also be
initialized without adding them to a disk group leaving the
```

```
  disks available for use as replacement disks.

  More than one disk or pattern may be entered at the prompt.
  Here are some disk selection examples:

  all:      all disks
  c3 c4t2:  all disks on both controller 3 and controller 4,
target 2
  c3t4d2:   a single disk (in the c#t#d# naming scheme)
  xyz_0:    a single disk (in the enclosure based naming scheme)
  xyz_:     all disks on the enclosure whose name is xyz

  disk#:        a single disk (in the new naming scheme)

  Select disk devices to add:
  [<pattern-list>,all,list,q,?]
```

<pattern-list> can be a single disk, or a series of disks and/or controllers (with optional targets). If *<pattern-list>* consists of multiple items, separate them using white space, for example:

```
c3t0d0 c3t1d0 c3t2d0 c3t3d0
```

specifies fours disks at separate target IDs on controller 3.

If you enter **`list`** at the prompt, the `vxdiskadm` program displays a list of the disks available to the system:

```
  DEVICE       DISK         GROUP        STATUS
  c2t4d0       -            -            LVM
  c2t5d0       -            -            LVM
  c2t6d0       -            -            LVM
  c3t0d0       mydg01       mydg         online
  c3t1d0       mydg03       mydg         online
  c3t2d0       mydg04       mydg         online
  c3t3d0       mydg05       mydg         online
  c3t8d0       mydg06       mydg         online
  c3t9d0       mydg07       mydg         online
  c3t10d0      mydg02       mydg         online
  c4t1d0       mydg08       mydg         online
  c4t2d0       TCd1-18238   TCg1-18238   online
  c4t13d0      -            -            online invalid
  c4t14d0      -            -            online
  .
  .
  .

  Select disk devices to add:
  [<pattern-list>,all,list,q,?]
```

The phrase `online invalid` in the `STATUS` line indicates that a disk has yet to be added or initialized for VxVM control. Disks that are listed as `online` with a disk name and disk group are already under VxVM control.

Enter the device name or pattern of the disks that you want to initialize at the prompt and press Return.

3 To continue with the operation, enter **y** (or press Return) at the following prompt:

```
Here are the disks selected.  Output format: [Device]

list of device names

Continue operation? [y,n,q,?] (default: y) y
```

4 At the following prompt, specify the disk group to which the disk should be added, or **none** to reserve the disks for future use:

```
You can choose to add these disks to an existing disk group, a
new disk group, or you can leave these disks available for use
by future add or replacement operations. To create a new disk
group, select a disk group name that does not yet exist. To
leave the disks available for future use, specify a disk group
name of "none".

Which disk group [<group>,none,list,q,?]
```

5 If you specified the name of a disk group that does not already exist, `vxdiskadm` prompts for confirmation that you really want to create this new disk group:

```
There is no active disk group named disk group name.

Create a new group named disk group name? [y,n,q,?]
(default: y)y
```

You are then prompted to confirm whether the disk group should support the Cross-platform Data Sharing (CDS) feature:

```
Create the disk group as a CDS disk group? [y,n,q,?]
(default: n)
```

If the new disk group may be moved between different operating system platforms, enter **y**. Otherwise, enter **n**.

6 At the following prompt, either press Return to accept the default disk name or enter **n** to allow you to define your own disk names:

```
Use default disk names for the disks? [y,n,q,?] (default: y)
```

7 When prompted whether the disks should become hot-relocation spares, enter **n** (or press Return):

```
Add disks as spare disks for disk group name? [y,n,q,?]
(default: n) n
```

8 When prompted whether to exclude the disks from hot-relocation use, enter **n** (or press Return).

```
Exclude disks from hot-relocation use? [y,n,q,?}
(default: n) n
```

9 You are next prompted to choose whether you want to add a site tag to the disks:

```
Add site tag to disks? [y,n,q,?] (default: n)
```

A site tag is usually applied to disk arrays or enclosures, and is not required unless you want to use the Remote Mirror feature. If you enter **y** to choose to add a site tag, you are prompted to the site name at step 11.

10 To continue with the operation, enter **y** (or press Return) at the following prompt:

```
The selected disks will be added to the disk group
disk group name with default disk names.
list of device names
Continue with operation? [y,n,q,?] (default: y) y
```

11 If you chose to tag the disks with a site in step 9, you are now prompted to enter the site name that should be applied to the disks in each enclosure:

```
The following disk(s):

list of device names

belong to enclosure(s):

list of enclosure names

Enter site tag for disks on enclosure enclosure_name
[<name>,q,?] site_name
```

12 If one or more disks already contains a file system, `vxdiskadm` asks if you are sure that want to destroy it. Enter **y** to confirm this:

```
The following disk device appears to contain a currently
unmounted file system.
list of device names


Are you sure you want to destroy these file systems
[y,n,q,?] (default: n) y
```

`vxdiskadm` asks you to confirm that the devices are to be reinitialized before proceeding:

```
Reinitialize these devices? [y,n,q,?] (default: n) y
VxVM INFO V-5-2-205 Initializing device device name.
```

13 You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable `hpdisk`-format disk:

```
Enter the desired format [cdsdisk,hpdisk,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

14 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 32768 blocks (32MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 32768)
```

vxdiskadm then proceeds to add the disks.

```
Adding disk device device name to disk group disk group name with
disk name disk name.
.
.
.
```

Note: To bring LVM disks under VxVM control, use the Migration Utilities. See the *Veritas Volume Manager Migration Guide* for details.

15 At the following prompt, indicate whether you want to continue to initialize more disks (**y**) or return to the vxdiskadm main menu (**n**):

```
Add or initialize other disks? [y,n,q,?] (default: n)
```

See "Displaying and changing default disk layout attributes" on page 97 for details of how to change the default layout that is used to initialize disks.

Reinitializing a disk

You can reinitialize a disk that has previously been initialized for use by VxVM by putting it under VxVM control as you would a new disk. See "Adding a disk to VxVM" on page 97 for details.

Caution: Reinitialization does not preserve data on the disk. If you want to reinitialize the disk, make sure that it does not contain data that should be preserved.

If the disk you want to add has previously been under LVM control, you can preserve the data it contains on a VxVM disk by the process of conversion (see the *Veritas Volume Manager Migration Guide* for more details).

Using vxdiskadd to place a disk under control of VxVM

As an alternative to vxdiskadm, you can use the vxdiskadd command to put a disk under VxVM control. For example, to initialize the second disk on the first controller, use the following command:

```
# vxdiskadd c0t1d0
```

The vxdiskadd command examines your disk to determine whether it has been initialized and also checks for disks that have been added to VxVM, and for other conditions.

Note: If you are adding an uninitialized disk, warning and error messages are displayed on the console during the `vxdiskadd` command. Ignore these messages. These messages should not appear after the disk has been fully initialized; the `vxdiskadd` command displays a success message when the initialization completes.

The interactive dialog for adding a disk using vxdiskadd is similar to that for `vxdiskadm`, described in "Adding a disk to VxVM" on page 97.

Rootability

Rootability indicates that the volumes containing the `root` file system and the system swap area are under VxVM control. Without rootability, VxVM is usually started *after* the operating system kernel has passed control to the initial user mode process at boot time. However, if the volume containing the `root` file system is under VxVM control, the kernel starts portions of VxVM *before* starting the first user mode process.

Under HP-UX, a bootable root disk contains a Logical Interchange Format (LIF) area. The LIF LABEL record in the LIF area contains information about the starting block number, and the length of the volumes that contain the `stand` and `root` file systems and the system swap area. When a VxVM root disk is made bootable, the LIF LABEL record is initialized with volume extent information for the `stand`, `root`, `swap`, and `dump` (if present) volumes.

See "Setting up a VxVM root disk and mirror" on page 104 for details of how to configure a bootable VxVM root disk from an existing LVM root disk.

Note: From the AR0902 release of HP-UX 11i onward, you can choose to configure either a VxVM root disk or an LVM root disk at install time. See the *HP-UX Installation and Configuration Guide* for more information.

See the chapter "Recovery from Boot Disk Failure" in the *Veritas Volume Manager Troubleshooting Guide*, for information on how to replace a failed boot disk.

VxVM root disk volume restrictions

Volumes on a bootable VxVM root disk have the following configuration restrictions:

- All volumes on the root disk must be in the disk group that you choose to be the `bootdg` disk group.
- The names of the volumes with entries in the LIF LABEL record must be `standvol`, `rootvol`, `swapvol`, and `dumpvol` (if present). The names of the volumes for other file systems on the root disk are generated by appending `vol` to the name of their mount point under `/`.
- Any volume with an entry in the LIF LABEL record must be contiguous. It can have only one subdisk, and it cannot span to another disk.
- The `rootvol` and `swapvol` volumes must have the special volume usage types `root` and `swap` respectively.
- Only the disk access types `auto` with format `hpdisk`, and `simple` are suitable for use as VxVM root disks, root disk mirrors, or as hot-relocation spares for such disks. An auto-configured `cdsdisk` format disk, which supports the Cross-platform Data Sharing (CDS) feature, cannot be used. The `vxcp_lvmroot` and `vxrootmir` commands automatically configure a suitable disk type on the physical disks that you specify are to be used as VxVM root disks and mirrors.
- The volumes on the root disk cannot use dirty region logging (DRL).

In addition, the size of the private region for disks in a VxVM boot disk group is limited to 1MB, rather than the usual default value of 32MB. This restriction is necessary to allow the boot loader to find the `/stand` file system during Maintenance Mode Boot.

Root disk mirrors

All the volumes on a VxVM root disk may be mirrored. The simplest way to achieve this is to mirror the VxVM root disk onto an identically sized or larger physical disk. If a mirror of the root disk must also be bootable, the restrictions listed in “Booting root volumes” on page 104 also apply to the mirror disk.

Note: If you mirror only selected volumes on the root disk and use spanning or striping to enhance performance, these mirrors are not bootable.

See “Setting up a VxVM root disk and mirror” on page 104 for details of how to create a mirror of a VxVM root disk.

Booting root volumes

Note: At boot time, the system firmware provides you with a short time period during which you can manually override the automatic boot process and select an alternate boot device. For information on how to boot your system from a device other than the primary or alternate boot devices, and how to change the primary and alternate boot devices, see the HP-UX documentation and the `boot`(1M), `pdc`(1M) and `isl`(1M) manual pages.

Before the kernel mounts the `root` file system, it determines if the boot disk is a rootable VxVM disk. If it is such a disk, the kernel passes control to its VxVM rootability code. This code extracts the starting block number and length of the `root` and `swap` volumes from the LIF LABEL record, builds temporary volume and disk configuration objects for these volumes, and then loads this configuration into the VxVM kernel driver. At this point, I/O can take place for these temporary `root` and `swap` volumes by referencing the device number set up by the rootability code.

When the kernel has passed control to the initial user procedure, the VxVM configuration daemon (`vxconfigd`) is started. `vxconfigd` reads the configuration of the volumes in the `bootdg` disk group and loads them into the kernel. The temporary `root` and `swap` volumes are then discarded. Further I/O for these volumes is performed using the VxVM configuration objects that were loaded into the kernel.

Setting up a VxVM root disk and mirror

Note: These procedures should be carried out at `init` level 1.

To set up a VxVM root disk and a bootable mirror of this disk, use the `vxcp_lvmroot` utility. This command initializes a specified physical disk as a VxVM root disk named `rootdisk##` (where `##` is the first number starting at `01` that creates a unique disk name), copies the contents of the volumes on the LVM root disk to the new VxVM root disk, optionally creates a mirror of the VxVM root disk on another specified physical disk, and make the VxVM root disk and its mirror (if any) bootable by HP-UX.

The following example shows how to set up a VxVM root disk on the physical disk `c0t4d0`:

```
# /etc/vx/bin/vxcp_lvmroot -b c0t4d0
```

Note: The `-b` option to `vxcp_lvmroot` uses the `setboot` command to define `c0t4d0` as the *primary* boot device. If this option is not specified, the primary boot device is not changed.

If the destination VxVM root disk is not big enough to accommodate the contents of the LVM root disk, you can use the `-R` option to specify a percentage by which to reduce the size of the file systems on the target disk. (This takes advantage of the fact that most of these file systems are usually nowhere near 100% full.) For example, to specify a size reduction of 30%, the following command would be used:

```
# /etc/vx/bin/vxcp_lvmroot -R 30 -v -b c0t4d0
```

The verbose option, `-v`, is specified to give an indication of the progress of the operation.

Caution: Only create a VxVM root disk if you also intend to mirror it. There is no benefit in having a non-mirrored VxVM root disk for its own sake.

The next example uses the same command and additionally specifies the `-m` option to set up a root mirror on disk `c1t1d0`:

```
# /etc/vx/bin/vxcp_lvmroot -m c1t1d0 -R 30 -v -b c0t4d0
```

In this example, the `-b` option to `vxcp_lvmroot` sets `c0t4d0` as the primary boot device and `c1t1d0` as the *alternate* boot device.

This command is equivalent to using `vxcp_lvmroot` to create the VxVM-rootable disk, and then using the `vxrootmir` command to create the mirror:

```
# /etc/vx/bin/vxcp_lvmroot -R 30 -v -b c0t4d0
# /etc/vx/bin/vxrootmir -v -b c1t1d0
```

The disk name assigned to the VxVM root disk mirror also uses the format `rootdisk##` with `##` set to the next available number.

Note: The target disk for a mirror that is added using the `vxrootmir` command must be large enough to accommodate the volumes from the VxVM root disk.

Once you have successfully rebooted the system from a VxVM root disk to `init` level 1, you can use the `vxdestroy_lvmroot` command to completely remove the original LVM root disk (and its associated LVM volume group), and re-use this disk as a mirror of the VxVM root disk, as shown in this example:

```
# /etc/vx/bin/vxdestroy_lvmroot -v c0t0d0
# /etc/vx/bin/vxrootmir -v -b c0t0d0
```

Note: You may want to keep the LVM root disk in case you ever need a boot disk that does not depend on VxVM being present on the system. However, this may require that you update the contents of the LVM root disk in parallel with changes that you make to the VxVM root disk. See "Creating an LVM root disk from a VxVM root disk" on page 106 for a description of how to create a bootable LVM root disk from the VxVM root disk.

For more information, see the `vxcp_lvmroot`(1M), `vxrootmir`(1M), `vxdestroy_lvmroot`(1M) and `vxres_lvmroot` (1M) manual pages.

Creating an LVM root disk from a VxVM root disk

Note: These procedures should be carried out at `init` level 1.

In some circumstances, it may be necessary to boot the system from an LVM root disk. If an LVM root disk is no longer available or an existing LVM root disk is out-of-date, you can use the `vxres_lvmroot` command to create an LVM root disk on a spare physical disk that is not currently under LVM or VxVM control. The contents of the volumes on the existing VxVM root disk are copied to the new LVM root disk, and the LVM disk is then made bootable. This operation does not remove the VxVM root disk or any mirrors of this disk, nor does it affect their bootability.

Note: The target disk must be large enough to accommodate the volumes from the VxVM root disk.

This example shows how to create an LVM root disk on physical disk `c0t1d0` after removing the existing LVM root disk configuration from that disk.

```
# /etc/vx/bin/vxdestroy_lvmroot -v c0t1d0
# /etc/vx/bin/vxres_lvmroot -v -b c0t1d0
```

The `-b` option to `vxres_lvmroot` sets `c0t1d0` as the primary boot device.

As these operations can take some time, the verbose option, `-v`, is specified to indicate how far the operation has progressed.

For more information, see the `vxres_lvmroot` (1M) manual page.

Adding swap volumes to a VxVM rootable system

To add a swap volume to an HP-UX system with a VxVM root disk

1. Initialize the disk that is to be used to hold the swap volume (for example, `c2t5d0`), and add it to the boot disk group with the disk media name "`swapdisk`":

   ```
   # /etc/vx/bin/vxdisksetup -i c2t5d0
   # vxdg -g bootdg adddisk swapdisk=c2t5d0
   ```

2. Create a VxVM volume on `swapdisk` (with a size of 4 gigabytes in this example):

   ```
   # vxassist -g bootdg -U swap make swapvol1 4g dm:swapdisk
   ```

 In this example, the size of the volume is 4 gigabytes.

3. Add the volume to the `/etc/fstab` file, and enable the volume as a swap device.

   ```
   # echo "/dev/vx/dsk/bootdg/swapvol1  - swap  defaults  0 0" \
     >> /etc/fstab
   # swapon -a
   ```

4. View the changed swap configuration:

   ```
   # swapinfo
   ```

Adding persistent dump volumes to a VxVM rootable system

A persistent dump volume is used when creating crash dumps, which are eventually saved in the `/var/adm/crash` directory. A maximum of ten VxVM volumes can be configured as persistent dump volumes.

To add a persistent dump volume to an HP-UX system with a VxVM root disk

1. Create the VxVM volume that is to be used as the dump volume in the boot disk group:

   ```
   # vxassist -g bootdg make dumpvol 5g
   ```

 In this example, the size of the volume is 5 gigabytes.

2. Display the configuration of the volume:

   ```
   # vxprint -g bootdg
   ```

3. Display the initial crash dump configuration:

   ```
   # crashconf -v
   ```

4. Add the volume as a persistent dump device to the crash dump configuration:

   ```
   # crashconf -s /dev/vx/dsk/bootdg/dumpvol
   ```

5. Display the new crash dump configuration:

   ```
   # crashconf -v
   ```

Removing a persistent dump volume

Caution: The system will not boot correctly if you delete a dump volume without first removing it from the crash dump configuration.

Use this procedure to remove a dump volume from the crash dump configuration.

To remove a persistent dump volume

1. Run the following command to remove a VxVM volume that is being used as a dump volume from the crash dump configuration:

   ```
   # crashconf -ds /dev/vx/dsk/bootdg/dumpvol
   ```

 In this example, the dump volume is named `dumpvol` in the boot disk group.

2. Display the new crash dump configuration:

   ```
   # crashconf -v
   ```

 You can now remove the volume if required.

Dynamic LUN expansion

Note: A license is required to use the dynamic LUN expansion feature.

The following form of the `vxdisk` command can be used to make VxVM aware of the new size of a virtual disk device that has been resized:

```
# vxdisk [-f] [-g diskgroup] resize {accessname|medianame} \
  [length=value]
```

The device must have a SCSI interface that is presented by a smart switch, smart array or RAID controller. Following a resize operation to increase the length that is defined for a device, additional disk space on the device is available for allocation. You can optionally specify the new size by using the `length` attribute.

If a disk media name rather than a disk access name is specified, the disk group must either be specified using the `-g` option or the default disk group will be used. If the default disk group has not been set up, an error message will be generated.

This facility is provided to support dynamic LUN expansion by updating disk headers and other VxVM structures to match a new LUN size. It does not resize the LUN itself.

Any volumes on the device should only be grown after the device itself has first been grown. Otherwise, storage other than the device may be used to grow the volumes, or the volume resize may fail if no free storage is available.

Resizing should only be performed on devices that preserve data. Consult the array documentation to verify that data preservation is supported and has been qualified. The operation also requires that only storage at the end of the LUN is affected. Data at the beginning of the LUN must not be altered. No attempt is made to verify the validity of pre-existing data on the LUN. The operation should be performed on the host where the disk group is imported (or on the master node for a cluster-shared disk group).

Resizing of LUNs that are not part of a disk group is not supported.It is not possible to resize LUNs that are in the boot disk group (aliased as `bootdg`), in a deported disk group, or that are offline, uninitialized, being reinitialized, or in an error state.

Caution: Do not perform this operation when replacing a physical disk with a disk of a different size as data is not preserved.

Before reducing the size of a device, any volumes on the device should first be reduced in size or moved off the device. By default, the resize fails if any subdisks would be disabled as a result of their being removed in whole or in part during a shrink operation.

If the device that is being resized has the only valid configuration copy for a disk group, the `-f` option may be specified to forcibly resize the device.

Resizing a device that contains the only valid configuration copy for a disk group can result in data loss if a system crash occurs during the resize.

Resizing a virtual disk device is a non-transactional operation outside the control of VxVM. This means that the resize command may have to be re-issued following a system crash. In addition, a system crash may leave the private region on the device in an unusable state. If this occurs, the disk must be reinitialized, reattached to the disk group, and its data resynchronized or recovered from a backup.

Removing disks

Note: You must disable a disk group as described in "Disabling a disk group" on page 207 before you can remove the last disk in that group. Alternatively, you can destroy the disk group as described in "Destroying a disk group" on page 208.

You can remove a disk from a system and move it to another system if the disk is failing or has failed.

To prepare your system for the removal of the disk

1. Stop all activity by applications to volumes that are configured on the disk that is to be removed. Unmount file systems and shut down databases that are configured on the volumes.
2. Use the following command to stop the volumes:

   ```
   # vxvol [-g diskgroup] stop volume1 volume2 ...
   ```

3. Move the volumes to other disks or back up the volumes. To move a volume, use `vxdiskadm` to mirror the volume on one or more disks, then remove the original copy of the volume. If the volumes are no longer needed, they can be removed instead of moved.
4. Check that any data on the disk has either been moved to other disks or is no longer needed.

To remove the disk from its disk group

1. Select menu item `2 (Remove a disk)` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk name of the disk to be removed:

   ```
   Remove a disk
   Menu: VolumeManager/Disk/RemoveDisk


   Use this operation to remove a disk from a disk group. This
   operation takes a disk name as input. This is the same name
   that you gave to the disk when you added the disk to the disk
   group.

   Enter disk name [<disk>,list,q,?] mydg01
   ```

3. If there are any volumes on the disk, VxVM asks you whether they should be evacuated from the disk. If you wish to keep the volumes, answer **y**. Otherwise, answer **n**.
4. At the following verification prompt, press Return to continue:

   ```
   VxVM NOTICE V-5-2-284 Requested operation is to remove disk
   mydg01 from group mydg.
   ```

```
Continue with operation? [y,n,q,?] (default: y)
```

The `vxdiskadm` utility removes the disk from the disk group and displays the following success message:

```
VxVM INFO V-5-2-268 Removal of disk mydg01 is complete.
```

You can now remove the disk or leave it on your system as a replacement.

5 At the following prompt, indicate whether you want to remove other disks (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Remove another disk? [y,n,q,?] (default: n)
```

Removing a disk with subdisks

You can remove a disk on which some subdisks are defined. For example, you can consolidate all the volumes onto one disk. If you use the `vxdiskadm` program to remove a disk, you can choose to move volumes off that disk. To do this, run the `vxdiskadm` program and select item `2 (Remove a disk)` from the main menu.

If the disk is used by some subdisks, the following message is displayed:

```
VxVM ERROR V-5-2-369 The following volumes currently use part of
disk mydg02:

home usrvol

Volumes must be moved from mydg02 before it can be removed.

Move volumes to other disks? [y,n,q,?] (default: n)
```

If you choose **y**, then all subdisks are moved off the disk, if possible. Some subdisks are not movable. A subdisk may not be movable for one of the following reasons:

- There is not enough space on the remaining disks in the subdisk's disk group.
- Plexes or striped subdisks cannot be allocated on different disks from existing plexes or striped subdisks in the volume.

If the `vxdiskadm` program cannot move some subdisks, remove some plexes from some disks to free more space before proceeding with the disk removal operation. See "Removing a volume" on page 290 and "Taking plexes offline" on page 230 for information on how to remove volumes and plexes.

Removing a disk with no subdisks

To remove a disk that contains no subdisks from its disk group, run the `vxdiskadm` program and select item `2 (Remove a disk)` from the main menu, and respond to the prompts as shown in this example to remove `mydg02`:

```
Enter disk name [<disk>,list,q,?] mydg02

VxVM NOTICE V-5-2-284 Requested operation is to remove disk
mydg02 from group mydg.

Continue with operation? [y,n,q,?] (default: y) y
VxVM INFO V-5-2-268 Removal of disk mydg02 is complete.
Clobber disk headers? [y,n,q,?] (default: n) y
```

Enter **y** to remove the disk completely from VxVM control. If you do not want to remove the disk completely from VxVM control, enter **n**.

Removing a disk from VxVM control

After removing a disk from a disk group, you can permanently remove it from Veritas Volume Manager control by running the `vxdiskunsetup` command:

```
# /usr/lib/vxvm/bin/vxdiskunsetup c#t#d#
```

Caution: The `vxdiskunsetup` command removes a disk from Veritas Volume Manager control by erasing the VxVM metadata on the disk. To prevent data loss, any data on the disk should first be evacuated from the disk. The `vxdiskunsetup` command should only be used by a system administrator who is trained and knowledgeable about Veritas Volume Manager.

Removing and replacing disks

Note: A replacement disk should have the same disk geometry as the disk that failed. That is, the replacement disk should have the same bytes per sector, sectors per track, tracks per cylinder and sectors per cylinder, same number of cylinders, and the same number of accessible cylinders.

If failures are starting to occur on a disk, but the disk has not yet failed completely, you can replace the disk. This involves detaching the failed or failing disk from its disk group, followed by replacing the failed or failing disk with a new one. Replacing the disk can be postponed until a later date if necessary.

To replace a disk

1. Select menu item `3 (Remove a disk for replacement)` from the `vxdiskadm` main menu.
2. At the following prompt, enter the name of the disk to be replaced (or enter **`list`** for a list of disks):

```
Remove a disk for replacement
Menu: VolumeManager/Disk/RemoveForReplace

Use this menu operation to remove a physical disk from a disk
group, while retaining the disk name. This changes the state
for the disk name to a removed disk. If there are any
initialized disks that are not part of a disk group, you will
be given the option of using one of these disks as a
replacement.

Enter disk name [<disk>,list,q,?] mydg02
```

3. When you select a disk to remove for replacement, all volumes that are affected by the operation are displayed, for example:

```
VxVM NOTICE V-5-2-371 The following volumes will lose mirrors
as a result of this operation:

home src

No data on these volumes will be lost.

The following volumes are in use, and will be disabled as a
result of this operation:

mkting

Any applications using these volumes will fail future
accesses. These volumes will require restoration from backup.

Are you sure you want do this? [y,n,q,?] (default: n)
```

 To remove the disk, causing the named volumes to be disabled and data to be lost when the disk is replaced, enter **`y`** or press Return.

 To abandon removal of the disk, and back up or move the data associated with the volumes that would otherwise be disabled, enter **`n`** or **`q`** and press Return.

 For example, to move the volume `mkting` to a disk other than `mydg02`, use this command:

```
# vxassist move mkting !mydg02
```

 After backing up or moving the data in the volumes, start again from step 1 above.

4. At the following prompt, either select the device name of the replacement disk (from the list provided), press Return to choose the default disk, or enter **`none`** if you are going to replace the physical disk:

```
The following devices are available as replacements:
c0t1d0

You can choose one of these disks now, to replace mydg02.
Select "none" if you do not wish to select a replacement disk.

Choose a device, or select "none"
[<device>,none,q,?] (default: c0t1d0)
```

Note: Do not choose the old disk drive as a replacement even though it appears in the selection list. If necessary, you can choose to initialize a new disk.

If you enter **none** because you intend to replace the physical disk, see the section "Replacing a failed or removed disk" on page 115.

5 If you chose to replace the disk in step 4, press Return at the following prompt to confirm this:

```
VxVM NOTICE V-5-2-285 Requested operation is to remove mydg02
from group mydg. The removed disk will be replaced with disk
device c0t1d0.

Continue with operation? [y,n,q,?] (default: y)
```

`vxdiskadm` displays the following messages to indicate that the original disk is being removed:

```
VxVM NOTICE V-5-2-265 Removal of disk mydg02 completed
successfully.
VxVM NOTICE V-5-2-260 Proceeding to replace mydg02 with device
c0t1d0.
```

6 You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable `hpdisk`-format disk:

```
Enter the desired format [cdsdisk,hpdisk,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

7 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 32768 blocks (32 MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 32768)
```

8 If one of more mirror plexes were moved from the disk, you are now prompted whether FastResync should be used to resynchronize the plexes:

```
Use FMR for plex resync? [y,n,q,?] (default: n) y
vxdiskadm displays the following success message:
```

```
VxVM NOTICE V-5-2-158 Disk replacement completed successfully.
```

9 At the following prompt, indicate whether you want to remove another disk (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Remove another disk? [y,n,q,?] (default: n)
```

Note: If removing a disk causes one or more volumes to be disabled, see the section, "Restarting a Disabled Volume" in the chapter "Recovery from Hardware Failure" in the *Veritas Volume Manager Troubleshooting Guide*, for information on how to restart a disabled volume so that you can restore its data from a backup.

If you wish to move hot-relocate subdisks back to a replacement disk, see "Configuring hot-relocation to use only spare disks" on page 390.

Replacing a failed or removed disk

Note: You may need to run commands that are specific to the operating system or disk array when replacing a physical disk.

To specify a disk that has replaced a failed or removed disk

1 Select menu item `4 (Replace a failed or removed disk)` from the `vxdiskadm` main menu.

2 At the following prompt, enter the name of the disk to be replaced (or enter **`list`** for a list of disks):

```
Replace a failed or removed disk
Menu: VolumeManager/Disk/ReplaceDisk

VxVM INFO V-5-2-479 Use this menu operation to specify a
replacement disk for a disk that you removed with the "Remove
a disk for replacement" menu operation, or that failed during
use. You will be prompted for a disk name to replace and a disk
device to use as a replacement.
You can choose an uninitialized disk, in which case the disk
will be initialized, or you can choose a disk that you have
already initialized using the Add or initialize a disk menu
operation.

Select a removed or failed disk [<disk>,list,q,?] mydg02
```

3 The `vxdiskadm` program displays the device names of the disk devices available for use as replacement disks. Your system may use a device name that differs from the examples. Enter the device name of the disk or press Return to select the default device:

```
The following devices are available as replacements:
```

```
c0t1d0 c1t1d0

You can choose one of these disks to replace mydg02.
Choose "none" to initialize another disk to replace mydg02.

Choose a device, or select "none"
[<device>,none,q,?] (default: c0t1d0)
```

4 Depending on whether the replacement disk was previously initialized, perform the appropriate step from the following:

- If the disk has not previously been initialized, press Return at the following prompt to replace the disk:

```
VxVM INFO V-5-2-378 The requested operation is to initialize
disk device c0t1d0 and to then use that device to
replace the removed or failed disk mydg02 in disk group mydg.
Continue with operation? [y,n,q,?] (default: y)
```

- If the disk has already been initialized, press Return at the following prompt to replace the disk:

```
VxVM INFO V-5-2-382 The requested operation is to use the
initialized device c0t1d0 to replace the removed or
failed disk mydg02 in disk group mydg.
Continue with operation? [y,n,q,?] (default: y)
```

5 You can now choose whether the disk is to be formatted as a CDS disk that is portable between different operating systems, or as a non-portable `hpdisk`-format disk:

```
Enter the desired format [cdsdisk,hpdisk,q,?]
(default: cdsdisk)
```

Enter the format that is appropriate for your needs. In most cases, this is the default format, `cdsdisk`.

6 At the following prompt, `vxdiskadm` asks if you want to use the default private region size of 32768 blocks (32 MB). Press Return to confirm that you want to use the default value, or enter a different value. (The maximum value that you can specify is 524288 blocks.)

```
Enter desired private region length [<privlen>,q,?]
(default: 32768)
```

7 The `vxdiskadm` program then proceeds to replace the disk, and returns the following message on success:

```
VxVM NOTICE V-5-2-158 Disk replacement completed successfully.
```

At the following prompt, indicate whether you want to replace another disk (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Replace another disk? [y,n,q,?] (default: n)
```

8 After using the `vxdiskadm` command to replace one or more failed disks in a VxVM cluster, run the following command on all the cluster nodes:

```
# vxdctl enable
```

Then run the following command on the master node:

```
# vxreattach -r accesname
```

where *accessname* is the disk access name (such as `c0t1d0`). This initiates the recovery of all the volumes on the disks.

Alternatively, halt the cluster, reboot all the cluster nodes, and restart the cluster.

If you do not perform this step for a cluster, the nodes may not be able to see the disks, and the error `Device path not valid` will be displayed.

Enabling a disk

If you move a disk from one system to another during normal system operation, VxVM does not recognize the disk automatically. The enable disk task enables VxVM to identify the disk and to determine if this disk is part of a disk group. Also, this task re-enables access to a disk that was disabled by either the disk group deport task or the disk device disable (offline) task.

To enable a disk

1 Select menu item `9 (Enable (online) a disk device)` from the `vxdiskadm` main menu.

2 At the following prompt, enter the device name of the disk to be enabled (or enter `list` for a list of devices):

```
Enable (online) a disk device
Menu: VolumeManager/Disk/OnlineDisk

VxVM INFO V-5-2-998 Use this operation to enable access to a
disk that was disabled with the "Disable (offline) a disk
device" operation.

You can also use this operation to re-scan a disk that may have
been changed outside of the Volume Manager. For example, if a
disk is shared between two systems, the Volume Manager running
on the other system may have changed the disk. If so, you can
use this operation to re-scan the disk.

NOTE: Many vxdiskadm operations re-scan disks without user
intervention. This will eliminate most needs to online a
disk directly, except when the disk is directly offlined.

Select a disk device to enable [<address>,list,q,?]
c0t2d0
```

vxdiskadm enables the specified device.

3 At the following prompt, indicate whether you want to enable another device (**y**) or return to the vxdiskadm main menu (**n**):

```
Enable another device? [y,n,q,?] (default: n)
```

Taking a disk offline

There are instances when you must take a disk offline. If a disk is corrupted, you must disable the disk before removing it. You must also disable a disk before moving the physical disk device to another location to be connected to another system.

Note: Taking a disk offline is only useful on systems that support *hot-swap* removal and insertion of disks without needing to shut down and reboot the system.

To take a disk offline

1 Select menu item `10 (Disable (offline) a disk device)` from the vxdiskadm main menu.

2 At the following prompt, enter the address of the disk you want to disable:

```
Disable (offline) a disk device
Menu: VolumeManager/Disk/OfflineDisk

VxVM INFO V-5-2-474 Use this menu operation to disable all
access to a disk device by the Volume Manager. This operation
can be applied only to disks that are not currently in a disk
group. Use this operation if you intend to remove a disk from
a system without rebooting.

NOTE: Many systems do not support disks that can be removed
from a system during normal operation. On such systems, the
offline operation is seldom useful.
Select a disk device to disable [<address>,list,q,?]
c0t2d0
```

The vxdiskadm program disables the specified disk.

3 At the following prompt, indicate whether you want to disable another device (**y**) or return to the vxdiskadm main menu (**n**):

```
Disable another device? [y,n,q,?] (default: n)
```

Renaming a disk

If you do not specify a VM disk name, VxVM gives the disk a default name when you add the disk to VxVM control. The VM disk name is used by VxVM to identify the location of the disk or the disk type. To change the disk name to reflect a change of use or ownership, use the following command:

```
# vxedit [-g diskgroup] rename old_diskname new_diskname
```

For example, you might want to rename disk `mydg03`, as shown in the following output from `vxdisk list`, to `mydg02`: #

```
# vxdisk list
DEVICE      TYPE           DISK        GROUP       STATUS
c0t0d0      auto:hpdisk    mydg01      mydg        online
c1t0d0      auto:hpdisk    mydg03      mydg        online
c1t1d0      auto:hpdisk    -           -           online
```

You would use the following command to rename the disk.

```
# vxedit -g mydg rename mydg03 mydg02
```

To confirm that the name change took place, use the `vxdisk list` command again:

```
# vxdisk list
DEVICE      TYPE           DISK        GROUP       STATUS
c0t0d0      auto:hpdisk    mydg01      mydg        online
c1t0d0      auto:hpdisk    mydg02      mydg        online
c1t1d0      auto:hpdisk    -           -           online
```

Note: By default, VxVM names subdisk objects after the VM disk on which they are located. Renaming a VM disk does not automatically rename the subdisks on that disk.

Reserving disks

By default, the `vxassist` command allocates space from any disk that has free space. You can reserve a set of disks for special purposes, such as to avoid general use of a particularly slow or a particularly fast disk.

To reserve a disk for special purposes, use the following command:

```
# vxedit [-g diskgroup] set reserve=on diskname
```

After you enter this command, the `vxassist` program does not allocate space from the selected disk unless that disk is specifically mentioned on the `vxassist` command line. For example, if mydg03 is reserved, use the following command:

```
# vxassist [-g diskgroup] make vol03 20m mydg03
```

The vxassist command overrides the reservation and creates a 20 megabyte volume on mydg03. However, the command:

```
# vxassist -g mydg make vol04 20m
```

does not use mydg03, even if there is no free space on any other disk.

To turn off reservation of a disk, use the following command:

```
# vxedit [-g diskgroup] set reserve=off diskname
```

See the vxedit(1M) manual page for more information.

Displaying disk information

Before you use a disk, you need to know if it has been initialized and placed under VxVM control. You also need to know if the disk is part of a disk group, because you cannot create volumes on a disk that is not part of a disk group. The vxdisk list command displays device names for all recognized disks, the disk names, the disk group names associated with each disk, and the status of each disk.

To display information on all disks that are known to VxVM, use the following command:

```
# vxdisk list
```

VxVM returns a display similar to the following:

```
DEVICE      TYPE            DISK        GROUP     STATUS
c0t0d0      auto:hpdisk     mydg04      mydg      online
c1t0d0      auto:hpdisk     mydg03      mydg      online
c1t1d0      auto:hpdisk     -           -         online invalid
enc0_2      auto:hpdisk     mydg02      mydg      online
enc0_3      auto:hpdisk     mydg05      mydg      online
enc0_0      auto:hpdisk     -           -         online
enc0_1      auto:hpdisk     -           -         online
```

The phrase online invalid in the STATUS line indicates that a disk has not yet been added to VxVM control. These disks may or may not have been initialized by VxVM previously. Disks that are listed as online are already under VxVM control.

VxVM cannot access stale device entries in the /dev/disk and /dev/rdisk directories. I/O cannot be performed to such devices, which are shown as being in the error state.

To display details on a particular disk that is defined to VxVM, use the following command:

```
# vxdisk [-v] list diskname
```

The -v option causes the command to additionally list all tags and tag values that are defined for the disk. Without this option, no tags are displayed.

Displaying disk information with vxdiskadm

Displaying disk information shows you which disks are initialized, to which disk groups they belong, and the disk status. The `list` command displays device names for all recognized disks, the disk names, the disk group names associated with each disk, and the status of each disk.

To display disk information

1. Start the `vxdiskadm` program, and select `list (List disk information)` from the main menu.
2. At the following display, enter the address of the disk you want to see, or enter **`all`** for a list of all disks:

   ```
   List disk information
   Menu: VolumeManager/Disk/ListDisk

   VxVM INFO V-5-2-475 Use this menu operation to display a list of
   disks. You can also choose to list detailed information about
   the disk at a specific disk device address.

   Enter disk device or "all" [<address>,all,q,?] (default: all)
   ```

 - If you enter **`all`**, VxVM displays the device name, disk name, group, and status.
 - If you enter the address of the device for which you want information, complete disk information (including the device name, the type of disk, and information about the public and private areas of the disk) is displayed.

 Once you have examined this information, press Return to return to the main menu.

Controlling Powerfail Timeout

Powerfail Timeout is an attribute of a SCSI disk connected to an HP-UX host. This is used to detect and handle I/O on non-responding disks.

See the `pfto`(7) man page.

VxVM uses this mechanism in its Powerfail Timeout (pfto) feature. You can specify a timeout value for individual VxVM disks using the `vxdisk` command. If a disk fails to respond in the specified timeout period, the driver receives a timer interrupt.

You can set the PFTO values on a disk or a set of disks within a disk group using the CLI. PFTO helps in preventing system hangs due to non-responding disks. However, in some circumstances, it is critical to retry the I/O until it succeeds, regardless of how long it takes. In those cases, you can disable PFTO. By default, the use of PFTO is enabled.

Setting the PFTO values

To set the PFTO value on a disk, use the following command:

```
$ vxdisk -g dg_name set disk_name pfto=value
```

For example, to set the PFTO value of 50sec on the disk c5t0d6:

```
$ vxdisk -g testdg set c5t0d6 pfto=50
```

To set the PFTO value on a disk group, use the following command:

```
$ vxpfto -g dg_name -t value
```

For example, to set the PFTO value to 50 seconds on all disks in the diskgroup testdg:

```
$ vxpfto -g testdg -t 50
```

For more details, see the `vxpfto`(1M) and `vxdisk`(1M) manual pages.

Displaying the PFTO values

To display the PFTO value and whether PFTO is enabled or disabled for a disk, use one of the following commands:

```
$ vxprint -g dg_name -l disk_name
$ vxdisk -g dg_name list disk_name
```

The output shows the `pftostate` field, which indicates whether PFTO is enabled or disabled. The `timeout` field shows the PFTO timeout value. For example, the output for the `c5t0d6` disk shows:

```
Device:    c5t0d6
devicetag: c5t0d6
...
timeout:   30
pftostate: disabled
...
```

Enabling or disabling PFTO

To enable or disable PFTO on a disk, use the following command:

```
$ vxdisk -g dg_name set disk_name pftostate={enabled|disabled}
```

For example, to disable PFTO on the disk c5t0d6:

```
$ vxdisk -g testdg set c5t0d6 pftostate=disabled
```

To enable or disable PFTO on a disk group, use the following command:

```
$ vxpfto -g dg_name -o pftostate={enabled|disabled}
```

For example, to disable PFTO on all disks in the diskgroup testdg:

```
$ vxpfto -g testdg -o pftostate=disabled
```

For more details, see the `vxpfto`(1M) and `vxdisk`(1M) manual pages.

Chapter 3

Administering dynamic multipathing (DMP)

The dynamic multipathing (DMP) feature of Veritas Volume Manager (VxVM) provides greater availability, reliability and performance by using path failover and load balancing. This feature is available for multiported disk arrays from various vendors.

DMP coexists with the native multipathing in HP-UX. For more information, see "DMP coexistence with HP-UX native multipathing" on page 130.

How DMP works

Multiported disk arrays can be connected to host systems through multiple paths. To detect the various paths to a disk, DMP uses a mechanism that is specific to each supported array type. DMP can also differentiate between different enclosures of a supported array type that are connected to the same host system.

See "Discovering and configuring newly added disk devices" on page 82 for a description of how to make newly added disk hardware known to a host system.

The multipathing policy used by DMP depends on the characteristics of the disk array:

- An *Active/Passive* array (A/P array) allows access to its LUNs (*logical units*; real disks or virtual disks created using hardware) via the *primary* (active) path on a single controller (also known as an *access port* or a *storage processor*) during normal operation.
 In *implicit failover mode* (or *autotrespass mode*), an A/P array automatically fails over by scheduling I/O to the *secondary* (passive) path on a separate controller if the primary path fails. This passive port is not used for I/O until the active port fails. In A/P arrays, path failover can occur for a single LUN if I/O fails on the primary path.

For Active/Passive arrays with *LUN group failover* (A/PG arrays), a group of LUNs that are connected through a controller is treated as a single failover entity. Unlike A/P arrays, failover occurs at the controller level, and not for individual LUNs. The primary and secondary controller are each connected to a separate group of LUNs. If a single LUN in the primary controller's LUN group fails, all LUNs in that group fail over to the secondary controller.

Active/Passive arrays in *explicit failover mode* (or *non-autotrespass mode*) are termed A/PF arrays. DMP issues the appropriate low-level command to make the LUNs fail over to the secondary path.

The paths of an Active/Passive array are not considered to be on different controllers when mirroring across controllers (for example, when creating a volume using `vxassist make` specified with the `mirror=ctlr` attribute).

A/P-C, A/PF-C and A/PG-C arrays are variants of the A/P, A/PF and A/PG array types that support concurrent I/O and load balancing by having multiple primary paths into a controller. This functionality is provided by a controller with multiple ports, or by the insertion of a SAN hub or switch between an array and a controller. Failover to the secondary (passive) path occurs only if all the active primary paths fail.

- An *Active/Active* disk array (A/A arrays) permits several paths to be used concurrently for I/O. Such arrays allow DMP to provide greater I/O throughput by balancing the I/O load uniformly across the multiple paths to the LUNs. In the event that one path fails, DMP automatically routes I/O over the other available paths.

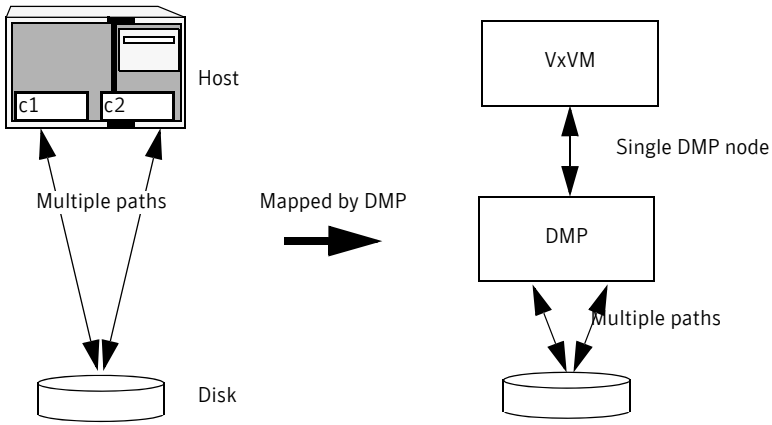
 A/A-A or Asymmetric Active/Active arrays can be accessed through secondary storage paths with little performance degradation. Usually an A/A-A array behaves like an A/P array rather than an A/A array. However, during failover, an A/A-A array behaves like an A/A array.

Note: An array support library (ASL) may define additional array types for the arrays that it supports.

VxVM uses *DMP metanodes* (*DMP nodes*) to access disk devices connected to the system. For each disk in a supported array, DMP maps one node to the set of paths that are connected to the disk. Additionally, DMP associates the appropriate multipathing policy for the disk array with the node. For disks in an unsupported array, DMP maps a separate node to each path that is connected to a disk. The raw and block devices for the nodes are created in the directories `/dev/vx/rdmp` and `/dev/vx/dmp` respectively.

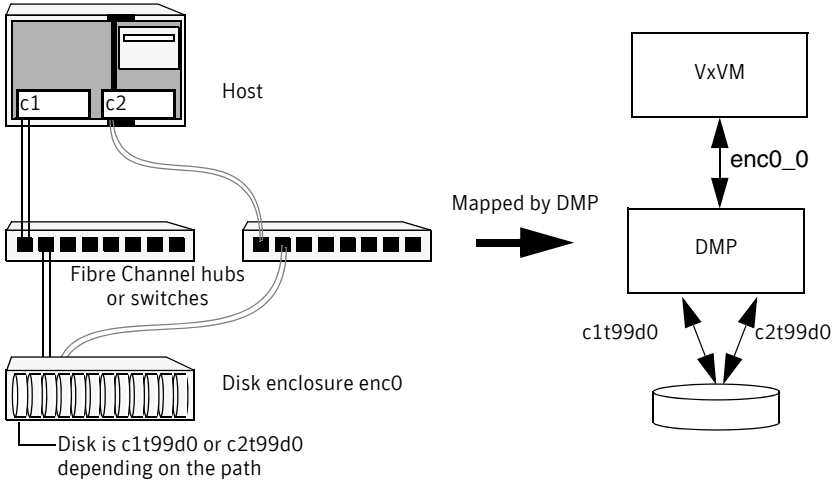
Figure 3-1 illustrates how DMP sets up a node for a disk in a supported disk array.

Figure 3-1 How DMP represents multiple physical paths to a disk as one node



As described in "Enclosure-based naming" on page 23, VxVM implements a disk device naming scheme that allows you to recognize to which array a disk belongs. Figure 3-2, shows an example where two paths, `c1t99d0` and `c2t99d0`, exist to a single disk in the enclosure, but VxVM uses the single DMP node, `enc0_0`, to access it.

Figure 3-2 Example of multipathing for a disk enclosure in a SAN environment



See “Changing the disk-naming scheme” on page 91 for details of how to change the naming scheme that VxVM uses for disk devices.

See “Discovering and configuring newly added disk devices” on page 82 for a description of how to make newly added disk hardware known to a host system.

How DMP monitors I/O on paths

In older releases of VxVM, DMP had one kernel daemon (`errord`) that performed error processing, and another (`restored`) that performed path restoration activities.

From release 5.0, DMP maintains a pool of kernel threads that are used to perform such tasks as error processing, path restoration, statistics collection, and SCSI request callbacks. The `vxdmpadm stat` command can be used to provide information about the threads. The names `errord` and `restored` have been retained for backward compatibility.

One kernel thread responds to I/O failures on a path by initiating a probe of the host bus adapter (HBA) that corresponds to the path. Another thread then takes the appropriate action according to the response from the HBA. The action taken can be to retry the I/O request on the path, or to fail the path and reschedule the I/O on an alternate path.

The restore kernel thread is woken periodically (typically every 5 minutes) to check the health of the paths, and to resume I/O on paths that have been restored. As some paths may suffer from intermittent failure, I/O is only resumed on a path if has remained healthy for a given period of time (by default, 5 minutes). DMP can be configured with different policies for checking the paths as described in “Configuring DMP path restoration policies” on page 160.

The statistics-gathering thread records the start and end time of each I/O request, and the number of I/O failures and retries on each path. DMP can be configured to use this information to prevent the SCSI driver being flooded by I/O requests. This feature is known as *I/O throttling*.

If an I/O request relates to a mirrored volume, VxVM specifies the FAILFAST flag. In such cases, DMP does not retry failed I/O requests on the path, and instead marks the disks on that path as having failed.

See “Path failover mechanism” on page 128 and “I/O throttling” on page 129 for more information about these features of DMP.

Path failover mechanism

The DMP feature of VxVM enhances system reliability when used with multiported disk arrays. In the event of the loss of a path to a disk array, DMP automatically selects the next available path for I/O requests without intervention from the administrator.

DMP is also informed when a connection is repaired or restored, and when you add or remove devices after the system has been fully booted (provided that the operating system recognizes the devices correctly).

If required, the response of DMP to I/O failure on a path can be tuned for the paths to individual arrays. DMP can be configured to time out an I/O request either after a given period of time has elapsed without the request succeeding, or after a given number of retries on a path have failed.

For information about how to configure the behavior of DMP in response to I/O failure on a path, see "Configuring the response to I/O failures" on page 156.

I/O throttling

If I/O throttling is enabled, and the number of outstanding I/O requests builds up on a path that has become less responsive, DMP can be configured to prevent new I/O requests being sent on the path either when the number of outstanding I/O requests has reached a given value, or a given time has elapsed since the last successful I/O request on the path. While throttling is applied to a path, the outstanding I/O requests on that path are scheduled on other available paths. The throttling is removed from the path if the HBA reports no error on the path, or if an outstanding I/O request on the path succeeds.

For information about how to configure I/O throttling on a path, see "Configuring the I/O throttling mechanism" on page 157.

Load balancing

By default, DMP uses a minimum queue length policy (`minimumq`) to provide load balancing across paths for Active/Active disk arrays. I/O is sent down the path that has the minimum number of outstanding I/O requests in the queue for a LUN.

For Active/Passive and Asymmetric Active/Active disk arrays, the default `round-robin` policy shares I/O equally between the available paths in a round-robin sequence. If the primary path fails, I/O is switched over to another available primary or secondary path. As the continuous transfer of ownership of LUNs from one controller to another results in severe I/O slowdown, load balancing across paths is not performed for Active/Passive disk arrays unless they support concurrent I/O.

You can use the `vxdmpadm` command to change the I/O policy for the paths to an enclosure or disk array.

See "Specifying the I/O policy" on page 147.

DMP coexistence with HP-UX native multipathing

The HP-UX 11i v3 release includes support for native multipathing, which can coexist with DMP. HP-UX native multipathing creates a persistent (*agile*) device in the `/dev/disk` and `/dev/rdisk` directories for each disk that can be accessed by one or more physical paths. To maintain backward compatibility, HP-UX also creates legacy devices in the `/dev/dsk` and `/dev/rdsk` directories.

VxVM recreates disk devices for all paths in the operating system's hardware device tree as DMP nodes in the `/dev/vx/dmp` and `/dev/vx/rdmp` directories, independently of the devices that are listed in the `/dev/dsk` and `/dev/rdsk` directories. VxVM uses a DMP node to represent a disk that can be accessed by one or more physical paths. DMP nodes are not used by the native multipathing feature of HP-UX.

VxVM commands display device names according the naming scheme that has been selected.

See "Changing the disk-naming scheme" on page 91.

By default, VxVM is configured to use DMP metanodes. If you want to use HP-UX native multipathing, you must add the HP-UX native multipathing metanodes as foreign devices.

See "Adding foreign devices" on page 89.

See "Migrating between DMP and HP-UX native multipathing" on page 130.

For instructions on administering native multipathing with `Base-VxVM` and `VxVM-Full`, please consult the *Release Notes* for VxVM 5.0 on 11i v3.

Migrating between DMP and HP-UX native multipathing

You can use the `vxddladm addforeign` and `vxddladm rmforeign` commands to migrate a system between using DMP and using HP-UX native multipathing. These procedures migrate all devices in the `/dev/disk` and `/dev/rdisk` directories.

Caution: Before migrating between DMP and HP-UX native multipathing, ensure that no applications are accessing VxVM volumes. Migration is not supported without first stopping any applications that are using the volumes.

To migrate from DMP to HP-UX native multipathing

1 Stop all the volumes in each disk group on the system:

```
# vxvol -g diskgroup stopall
```

2 Use the following commands to initiate the migration:

```
# vxddladm addforeign blockdir=/dev/disk chardir=/dev/rdisk
# vxconfigd -kr reset
```

3 Restart all the volumes in each disk group:

```
# vxvol -g diskgroup startall
```

The output from the `vxdisk list` command now shows only HP-UX native multipathing metanode names, for example:

```
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
disk155      auto:LVM        -            -            LVM
disk156      auto:LVM        -            -            LVM
disk224      auto:cdsdisk    -            -            online
disk225      auto:cdsdisk    -            -            online
disk226      auto:cdsdisk    -            -            online
disk227      auto:cdsdisk    -            -            online
disk228      auto:cdsdisk    -            -            online
disk229      auto:cdsdisk    -            -            online
```

When HP-UX native multipathing is configured, no DMP metanodes are configured for the devices in the `/dev/disk` and `/dev/rdisk` directories. As a result, the `vxdisk list` command only displays the names of the HP-UX native multipathing metanodes, and cannot display legacy names for the devices.

To migrate from HP-UX native multipathing to DMP

1 Stop all the volumes in each disk group on the system:

```
# vxvol -g diskgroup stopall
```

2 Use the following commands to initiate the migration:

```
# vxddladm rmforeign blockdir=/dev/disk chardir=/dev/rdisk
# vxconfigd -kr reset
```

3 Restart all the volumes in each disk group:

```
# vxvol -g diskgroup startall
```

The output from the `vxdisk list` command now shows DMP metanode names according to the current naming scheme. For example, under the `default` or `legacy` naming scheme, `vxdisk list` displays the devices as:

```
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
c2t0d0       auto:LVM        -            -            LVM
c3t2d0       auto:LVM        -            -            LVM
c89t0d0      auto:cdsdisk    -            -            online
c89t0d1      auto:cdsdisk    -            -            online
c89t0d2      auto:cdsdisk    -            -            online
c89t0d3      auto:cdsdisk    -            -            online
c89t0d4      auto:cdsdisk    -            -            online
c89t0d5      auto:cdsdisk    -            -            online
```

and under the `new` naming scheme as:

```
# vxdisk list
DEVICE       TYPE            DISK         GROUP        STATUS
disk155      auto:LVM        -            -            LVM
disk156      auto:LVM        -            -            LVM
disk224      auto:cdsdisk    -            -            online
disk225      auto:cdsdisk    -            -            online
disk226      auto:cdsdisk    -            -            online
disk227      auto:cdsdisk    -            -            online
disk228      auto:cdsdisk    -            -            online
disk229      auto:cdsdisk    -            -            online
```

See "Changing the disk-naming scheme" on page 91.

DMP in a clustered environment

Note: You need an additional license to use the cluster feature of VxVM.

In a clustered environment where Active/Passive type disk arrays are shared by multiple hosts, all nodes in the cluster must access the disk via the same physical path. Accessing a disk via multiple paths simultaneously can severely degrade I/O performance (sometimes referred to as the *ping-pong effect*). Path failover on a single cluster node is also coordinated across the cluster so that all the nodes continue to share the same physical path.

Prior to release 4.1 of VxVM, the clustering and DMP features could not handle automatic failback in A/P arrays when a path was restored, and did not support failback for explicit failover mode arrays. Failback could only be implemented manually by running the `vxdctl enable` command on each cluster node after the path failure had been corrected. In release 4.1, failback is now an automatic cluster-wide operation that is coordinated by the master node. Automatic failback in explicit failover mode arrays is also handled by issuing the appropriate low-level command. If required, this feature can be disabled by selecting the "no failback" option that is defined in the array policy module (APM) for an array.

Note: Support for automatic failback of an A/P array requires that an appropriate ASL (and APM, if required) is available for the array, and has been installed on the system. See "Administering the Device Discovery Layer" on page 85 and "Configuring array policy modules" on page 162.

For Active/Active type disk arrays, any disk can be simultaneously accessed through all available physical paths to it. In a clustered environment, the nodes do not all need to access a disk via the same physical path.

Enabling or disabling controllers with shared disk groups

Prior to release 5.0, VxVM did not allow enabling or disabling of paths or controllers connected to a disk that is part of a shared Veritas Volume Manager disk group. From VxVM 5.0 onward, such operations are supported on shared DMP nodes in a cluster.

Disabling and enabling multipathing for specific devices

You can use `vxdiskadm` menu options 17 and 18 to disable or enable multipathing. These menu options also allow you to exclude or exclude devices from the view of VxVM.

See "Disabling multipathing and making devices invisible to VxVM" on page 133.

See "Enabling multipathing and making devices visible to VxVM" on page 134.

Disabling multipathing and making devices invisible to VxVM

Note: Some of the operations described in this section require a reboot of the system.

1. Select menu task 17 (`Prevent multipathing/Suppress devices from VxVM's view`) from the `vxdiskadm` main menu to prevent a device from being multipathed by the VxVM DMP driver (`vxdmp`), or to exclude a device from the view of VxVM. You are prompted to confirm whether you want to continue.
2. Select the operation you want to perform from the displayed list:

```
1   Suppress all paths through a controller from VxVM's view
2   Suppress a path from VxVM's view
3   Suppress disks from VxVM's view by specifying a VID:PID
    combination
4   Suppress all but one paths to a disk
5   Prevent multipathing of all disks on a controller by VxVM
6   Prevent multipathing of a disk by VxVM
7   Prevent multipathing of disks by specifying a VID:PID
    combination
8   List currently suppressed/non-multipathed devices

?   Display help about menu
??  Display help about the menuing system
q   Exit from menus
```

Help text and examples are provided onscreen for all the menu items.

- Select option 1 to exclude all paths through the specified controller from the view of VxVM. These paths remain in the disabled state until the next reboot, or until the paths are re-included.
- Select option 2 to exclude specified paths from the view of VxVM.
- Select option 3 to exclude disks from the view of VxVM that match a specified Vendor ID and Product ID.
- Select option 4 to define a pathgroup for disks that are not multipathed by VxVM. (A pathgroup explicitly defines alternate paths to the same disk.) Only one path is made visible to VxVM.
- Select option 5 to disable multipathing for all disks on a specified controller.
- Select option 6 to disable multipathing for specified paths. The disks that correspond to a specified path are claimed in the OTHER_DISKS category and are not multipathed.
- Select option 7 to disable multipathing for disks that match a specified Vendor ID and Product ID. The disks that correspond to a specified Vendor ID and Product ID combination are claimed in the OTHER_DISKS category and are not multipathed.
- Select option 8 to list the devices that are currently suppressed or not multipathed.

Enabling multipathing and making devices visible to VxVM

Note: Some of the operations described in this section require a reboot of the system.

1. Select menu item 18 (`Allow multipathing/Unsuppress devices from VxVM's view`) from the `vxdiskadm` main menu to re-enable multipathing for a device, or to make a device visible to VxVM again. You are prompted to confirm whether you want to continue.
2. Select the operation you want to perform from the displayed list:

```
1   Unsuppress all paths through a controller from VxVM's view
2   Unsuppress a path from VxVM's view
3   Unsuppress disks from VxVM's view by specifying a VID:PID
    combination
4   Remove a pathgroup definition
5   Allow multipathing of all disks on a controller by VxVM
6   Allow multipathing of a disk by VxVM
7   Allow multipathing of disks by specifying a VID:PID
    combination
8   List currently suppressed/non-multipathed devices
```

```
?   Display help about menu
??  Display help about the menuing system
q   Exit from menus
```

- Select option 1 to make all paths through a specified controller visible to VxVM.
- Select option 2 to make specified paths visible to VxVM.
- Select option 3 to make disks visible to VxVM that match a specified Vendor ID and Product ID.
- Select option 4 to remove a pathgroup definition. (A pathgroup explicitly defines alternate paths to the same disk.) Once a pathgroup has been removed, all paths that were defined in that pathgroup become visible again.
- Select option 5 to enable multipathing for all disks that have paths through the specified controller.
- Select option 6 to enable multipathing for specified paths.
- Select option 7 to enable multipathing for disks that match a specified Vendor ID and Product ID.
- Select option 8 to list the devices that are currently suppressed or not multipathed.

Enabling and disabling I/O for controllers and storage processors

DMP allows you to turn off I/O for a controller or the array port of a storage processor so that you can perform administrative operations. This feature can be used for maintenance of HBA controllers on the host, or array ports that are attached to disk arrays supported by VxVM. I/O operations to the controller or array port can be turned back on after the maintenance task is completed. You can accomplish these operations using the `vxdmpadm` command provided with VxVM.

In Active/Active type disk arrays, VxVM uses a balanced path mechanism to schedule I/O to multipathed disks. As a result, I/O may go through any available path at any given point in time. For example, if a system has an Active/Active storage array, and you need to change an interface board that is connected to this disk array (if supported by the hardware), you can use the `vxdmpadm` command to list the controllers that are connected to the interface board. Disable the controllers to stop further I/O to the disks that are accessed through the interface board. You can then replace the board without causing disruption to any ongoing I/O to disks in the disk array.

In Active/Passive type disk arrays, VxVM schedules I/O to use the primary path until a failure is encountered. To change the interface card for an array port or an HBA controller card on the host (if supported by the hardware) that is connected to the disk array, disable I/O operations to the array port or to the HBA controller. This shifts all I/O over to an active secondary path or to an active primary path on another controller so that you can change the hardware.

After the operation is over, you can use `vxdmpadm` to re-enable the paths through the controllers.

See “Disabling I/O for paths, controllers or array ports” on page 153.

See “Enabling I/O for paths, controllers or array ports” on page 154.

See “Upgrading disk controller firmware” on page 154.

Note: From release 5.0 of VxVM, these operations are supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

Displaying DMP database information

You can use the vxdmpadm command to list DMP database information and perform other administrative tasks. This command allows you to list all controllers that are connected to disks, and other related information that is stored in the DMP database. You can use this information to locate system hardware, and to help you decide which controllers need to be enabled or disabled.

The vxdmpadm command also provides useful information such as disk array serial numbers, which DMP devices (disks) are connected to the disk array, and which paths are connected to a particular controller, enclosure or array port.

For more information, see "Administering DMP using vxdmpadm" on page 139.

Displaying the paths to a disk

The vxdisk command is used to display the multipathing information for a particular metadevice. The metadevice is a device representation of a particular physical disk having multiple physical paths from one of the system's HBA controllers. In VxVM, all the physical disks in the system are represented as metadevices with one or more physical paths.

You can use the vxdisk path command to display the relationships between the device paths, disk access names, disk media names and disk groups on a system as shown here:

```
# vxdisk path
SUBPATH      DANAME      DMNAME      GROUP       STATE
c1t0d0       c1t0d0      mydg01      mydg        ENABLED
c4t0d0       c1t0d0      mydg01      mydg        ENABLED
c1t1d0       c1t1d0      mydg02      mydg        ENABLED
c4t1d0       c1t1d0      mydg02      mydg        ENABLED
.
.
.
```

This shows that two paths exist to each of the two disks, mydg01 and mydg02, and also indicates that each disk is in the ENABLED state.

To view multipathing information for a particular metadevice, use the following command:

```
# vxdisk list devicename
```

For example, to view multipathing information for c1t0d3, use the following command:

```
# vxdisk list c1t0d3
```

Typical output from the vxdisk list command is as follows:

```
Device:    c1t0d3
```

```
devicetag: c1t0d3
type:      simple
hostid:    zort
disk:      name=mydg04 id=962923652.362193.zort
timeout:   30
group:     name=mydg id=962212937.1025.zort
info:      privoffset=128
flags:     online ready private autoconfig autoimport imported
pubpaths:  block=/dev/vx/dmp/c1t0d3
privpaths: char=/dev/vx/rdmp/c1t0d3
version:   2.1
iosize:    min=1024 (bytes) max=64 (blocks)
public:    slice=0 offset=1152 len=4101723
private:   slice=0 offset=128 len=1024
update:    time=962923719 seqno=0.7
headers:   0 248
configs:   count=1 len=727
logs:      count=1 len=110
Defined regions:
config   priv 000017-000247[000231]:copy=01 offset=000000
disabled
config   priv 000249-000744[000496]:copy=01 offset=000231
disabled
log      priv 000745-000854[000110]:copy=01 offset=000000
disabled
lockrgn  priv 000855-000919[000065]: part=00 offset=000000
Multipathing information:
numpaths:   2
c1t0d3 state=enabled  type=secondary
c4t1d3 state=disabled  type=primary
```

In the `Multipathing information` section of this output, the `numpaths` line shows that there are 2 paths to the device, and the following two lines show that the path to `c1t0d3` is active (`state=enabled`) and that the other path `c4t1d3` has failed (`state=disabled`).

The `type` field is shown for disks on Active/Passive type disk arrays such as the EMC CLARiiON, Hitachi HDS 9200 and 9500, Sun StorEdge 6xxx, and Sun StorEdge T3 array. This field indicates the *primary* and *secondary* paths to the disk.

The `type` field is not displayed for disks on Active/Active type disk arrays such as the EMC Symmetrix, Hitachi HDS 99xx and Sun StorEdge 99xx Series, and IBM ESS Series. Such arrays have no concept of primary and secondary paths.

Administering DMP using vxdmpadm

The vxdmpadm utility is a command line administrative interface to the DMP feature of VxVM. You can use the vxdmpadm utility to perform the following tasks.

- Retrieve the name of the DMP device corresponding to a particular path.
- Display the members of a LUN group.
- List all paths under a DMP device node, HBA controller or array port.
- Display information about the HBA controllers on the host.
- Display information about enclosures.
- Display information about array ports that are connected to the storage processors of enclosures.
- Display information about devices that are controlled by third-party multipathing drivers.
- Gather I/O statistics for a DMP node, enclosure, path or controller.
- Configure the attributes of the paths to an enclosure.
- Set the I/O policy that is used for the paths to an enclosure.
- Enable or disable I/O for a path, HBA controller or array port on the system.
- Upgrade disk controller firmware.
- Rename an enclosure.
- Configure how DMP responds to I/O request failures.
- Configure the I/O throttling mechanism.
- Control the operation of the DMP path restoration thread.

The following sections cover these tasks in detail along with sample output.

The vxdmpadm command can also be used to change the value of various DMP tunables. See “Changing the values of tunables” on page 474.

For more information about the vxdmpadm command, see the vxdmpadm(1M) manual page.

Retrieving information about a DMP node

The following command displays the DMP node that controls a particular physical path:

```
# vxdmpadm getdmpnode nodename=c3t2d1
```

The physical path is specified by argument to the `nodename` attribute, which must be a valid path listed in the `/dev/rdsk` directory.

The above command displays output such as the following:

```
NAME    STATE     ENCLR-TYPE    PATHS    ENBL    DSBL    ENCLR-NAME
=============================================================
c3t2d1  ENABLED   ACME          2        2       0       enc0
```

Use the `enclosure` attribute with `getdmpnode` to obtain a list of all DMP nodes for the specified enclosure.

```
# vxdmpadm getdmpnode enclosure=enc0
NAME    STATE     ENCLR-TYPE     PATHS   ENBL    DSBL    ENCLR-NAME
=============================================================
c2t1d0  ENABLED   ACME           2       2       0       enc0
c2t1d1  ENABLED   ACME           2       2       0       enc0
c2t1d2  ENABLED   ACME           2       2       0       enc0
c2t1d3  ENABLED   ACME           2       2       0       enc0
```

Displaying the members of a LUN group

The following command displays the DMP nodes that are in the same LUN group as a specified DMP node:

```
# vxdmpadm getlungroup dmpnodename=c11t0d10
```

The above command displays output such as the following:

```
NAME      STATE     ENCLR-TYPE  PATHS    ENBL    DSBL   ENCLR-NAME
=============================================================
c11t0d8   ENABLED   ACME        2        2       0      enc1
c11t0d9   ENABLED   ACME        2        2       0      enc1
c11t0d10  ENABLED   ACME        2        2       0      enc1
c11t0d11  ENABLED   ACME        2        2       0      enc1
```

Displaying paths controlled by a DMP node, controller or array port

The `vxdmpadm getsubpaths` command combined with the `dmpnodename` attribute displays all the paths to a LUN that are controlled by the specified DMP node name from the `/dev/vx/rdmp` directory:

```
# vxdmpadm getsubpaths dmpnodename=c2t66d0

NAME    STATE[A]    PATH-TYPE[M]CTLR-NAMEENCLR-TYPE ENCLR-NAME ATTRS
=============================================================================
c2t66d0 ENABLED(A)  PRIMARY      c2       ACME        enc0       -
c1t66d0 ENABLED     PRIMARY      c1       ACME        enc0       -
```

For A/A arrays, all enabled paths that are available for I/O are shown as `ENABLED(A)`.

For A/P arrays in which the I/O policy is set to `singleactive`, only one path is shown as `ENABLED(A)`. The other paths are enabled but not available for I/O. If the I/O policy is not set to `singleactive`, DMP can use a group of paths (all primary or all secondary) for I/O, which are shown as `ENABLED(A)`. See "Specifying the I/O policy" on page 147 for more information.

Paths that are in the DISABLED state are not available for I/O operations.

You can use `getsubpaths` to obtain information about all the paths that are connected to a particular HBA controller:

```
# vxdmpadm getsubpaths ctlr=c2

NAME    STATE[-]  PATH-TYPE[-] CTLR-NAME   ENCLR-TYPE  ENCLR-NAMEATTRS
==========================================================================
c2t1d0  ENABLED   PRIMARY      c2t1d0      ACME        enc0       -
c2t2d0  ENABLED   PRIMARY      c2t2d0      ACME        enc0       -
c2t3d0  ENABLED   SECONDARY    c2t3d0      ACME        enc0       -
c2t4d0  ENABLED   SECONDARY    c2t4d0      ACME        enc0       -
```

You can also use `getsubpaths` to obtain information about all the paths that are connected to a port on an array. The array port can be specified by the name of the enclosure and the array port ID, or by the worldwide name (WWN) identifier of the array port:

```
# vxdmpadm getsubpaths enclosure=HDS9500V0 portid=1A
# vxdmpadm getsubpaths pwwn=20:00:00:E0:8B:06:5F:19
```

Displaying information about controllers

The following command lists attributes of all HBA controllers on the system:

```
# vxdmpadm listctlr all

CTLR-NAME        ENCLR-TYPE         STATE          ENCLR-NAME
=============================================================
c1               OTHER              ENABLED        other0
c2               X1                 ENABLED        jbod0
c3               ACME               ENABLED        enc0
c4               ACME               ENABLED        enc0
```

This output shows that the controller `c1` is connected to disks that are not in any recognized DMP category as the enclosure type is `OTHER`.

The other controllers are connected to disks that are in recognized DMP categories.

All the controllers are in the ENABLED state which indicates that they are available for I/O operations.

The state DISABLED is used to indicate that controllers are unavailable for I/O operations. The unavailability can be due to a hardware failure or due to I/O

operations being disabled on that controller by using the `vxdmpadm disable` command.

This form of the command lists controllers belonging to a specified enclosure and enclosure type:

```
# vxdmpadm listctlr enclosure=enc0 type=ACME

CTLR-NAME        ENCLR-TYPE         STATE           ENCLR-NAME
=============================================================
c2               ACME               ENABLED         enc0
c3               ACME               ENABLED         enc0
```

Displaying information about enclosures

To display the attributes of a specified enclosure, including its enclosure type, enclosure serial number, status and array type, use the following command:

```
# vxdmpadm listenclosure enc0
ENCLR_NAME ENCLR_TYPE  ENCLR_SNO              STATUS   ARRAY_TYPE
==========================================================================
enc0       A3          60020f20000001a90000   CONNECTED  A/P
```

The following command lists attributes for all enclosures in a system:

```
# vxdmpadm listenclosure all
```

The following is example output from this command:

```
ENCLR_NAME ENCLR_TYPE    ENCLR_SNO             STATUS  ARRAY_TYPE
=============================================================
Disk         Disk        DISKS                 CONNECTED  Disk
ANA0         ACME        508002000001d660      CONNECTED  A/A
enc0         A3          60020f20000001a90000 CONNECTED  A/P
```

Displaying information about array ports

To display the attributes of an array port that is accessible via a path, DMP node or HBA controller, use one of the following commands:

```
# vxdmpadm getportids path=path-name
# vxdmpadm getportids dmpnodename=dmpnode-name
# vxdmpadm getportids ctlr=ctlr-name
```

The information displayed for an array port includes the name of its enclosure, and its ID and worldwide name (WWN) identifier.

The following form of the command displays information about all of the array ports within the specified enclosure:

```
# vxdmpadm getportids enclosure=enclr-name
```

The following example shows information about the array port that is accessible via DMP node `c2t66d0`:

```
# vxdmpadm getportids dmpnodename=c2t66d0
```

```
NAME        ENCLR-NAME  ARRAY-PORT-ID pWWN
============================================================
c2t66d0     HDS9500V0   1A            20:00:00:E0:8B:06:5F:19
```

Displaying information about TPD-controlled devices

The third-party driver (TPD) coexistence feature allows I/O that is controlled by third-party multipathing drivers to bypass DMP while retaining the monitoring capabilities of DMP. The following commands allow you to display the paths that DMP has discovered for a given TPD device, and the TPD device that corresponds to a given TPD-controlled node discovered by DMP:

```
# vxdmpadm getsubpaths tpdnodename=TPD_node_name
# vxdmpadm gettpdnode nodename=DMP_node_name
```

See "Changing device naming for TPD-controlled enclosures" on page 94 for information on how to select whether OS or TPD-based device names are displayed.

For example, consider the following disks in an EMC Symmetrix array controlled by PowerPath, which are known to DMP:

```
# vxdisk list

DEVICE          TYPE            DISK        GROUP     STATUS
emcpower10      auto:sliced     disk1       ppdg      online
emcpower11      auto:sliced     disk2       ppdg      online
emcpower12      auto:sliced     disk3       ppdg      online
emcpower13      auto:sliced     disk4       ppdg      online
emcpower14      auto:sliced     disk5       ppdg      online
emcpower15      auto:sliced     disk6       ppdg      online
emcpower16      auto:sliced     disk7       ppdg      online
emcpower17      auto:sliced     disk8       ppdg      online
emcpower18      auto:sliced     disk9       ppdg      online
emcpower19      auto:sliced     disk10      ppdg      online
```

The following command displays the paths that DMP has discovered, and which correspond to the PowerPath-controlled node, `emcpower10`:

```
# vxdmpadm getsubpaths tpdnodename=emcpower10

NAME    TPDNODENAME   PATH-TYPE[-]DMP-NODENAME  ENCLR-TYPE ENCLR-NAME
=====================================================================
c7t0d10 emcpower10s2  -            emcpower10    EMC        EMC0
c6t0d10 emcpower10s2  -            emcpower10    EMC        EMC0
```

Conversely, the next command displays information about the PowerPath node that corresponds to the path, `c7t0d10`, discovered by DMP:

```
# vxdmpadm gettpdnode nodename=c7t0d10

NAME          STATE     PATHS   ENCLR-TYPE    ENCLR-NAME
=============================================================
emcpower10s2  ENABLED   2       EMC           EMC0
```

Gathering and displaying I/O statistics

You can use the `vxdmpadm iostat` command to gather and display I/O statistics for a specified DMP node, enclosure, path or controller.

To enable the gathering of statistics, enter this command:

```
# vxdmpadm iostat start [memory=size]
```

To reset the I/O counters to zero, use this command:

```
# vxdmpadm iostat reset
```

The `memory` attribute can be used to limit the maximum amount of memory that is used to record I/O statistics for each CPU. The default limit is `32k` (32 kilobytes) per CPU.

To display the accumulated statistics at regular intervals, use the following command:

```
# vxdmpadm iostat show {all | dmpnodename=dmp-node | \
  enclosure=enclr-name | pathname=path-name | ctlr=ctlr-name} \
  [interval=seconds [count=N]]
```

This command displays I/O statistics for all controllers (`all`), or for a specified DMP node, enclosure, path or controller. The statistics displayed are the CPU usage and amount of memory per CPU used to accumulate statistics, the number of read and write operations, the number of kilobytes read and written, and the average time in milliseconds per kilobyte that is read or written.

The `interval` and `count` attributes may be used to specify the interval in seconds between displaying the I/O statistics, and the number of lines to be displayed. The actual interval may be smaller than the value specified if insufficient memory is available to record the statistics.

To disable the gathering of statistics, enter this command:

```
# vxdmpadm iostat stop
```

Examples of using the vxdmpadm iostat command

The follow is an example session using the `vxdmpadm iostat` command. The first command enables the gathering of I/O statistics:

```
# vxdmpadm iostat start
```

The next command displays the current statistics including the accumulated total numbers of read and write operations and kilobytes read and written, on all paths:

```
# vxdmpadm iostat show all
                    cpu usage = 7952us    per cpu memory = 8192b
                OPERATIONS            KBYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS   WRITES
c0t0d0       1088          0     557056          0  0.009542 0.000000
c2t118d0       87          0      44544          0  0.001194 0.000000
c3t118d0        0          0          0          0  0.000000 0.000000
c2t122d0       87          0      44544          0  0.007265 0.000000
c3t122d0        0          0          0          0  0.000000 0.000000
```

```
c2t115d0       87          0       44544          0  0.001200 0.000000
c3t115d0        0          0           0          0  0.000000 0.000000
c2t103d0       87          0       44544          0  0.007315 0.000000
c3t103d0        0          0           0          0  0.000000 0.000000
c2t102d0       87          0       44544          0  0.001132 0.000000
c3t102d0        0          0           0          0  0.000000 0.000000
c2t121d0       87          0       44544          0  0.000997 0.000000
c3t121d0        0          0           0          0  0.000000 0.000000
c2t112d0       87          0       44544          0  0.001559 0.000000
c3t112d0        0          0           0          0  0.000000 0.000000
c2t96d0        87          0       44544          0  0.007057 0.000000
c3t96d0         0          0           0          0  0.000000 0.000000
c2t106d0       87          0       44544          0  0.007247 0.000000
c3t106d0        0          0           0          0  0.000000 0.000000
c2t113d0       87          0       44544          0  0.007235 0.000000
c3t113d0        0          0           0          0  0.000000 0.000000
c2t119d0       87          0       44544          0  0.001390 0.000000
c3t119d0        0          0           0          0  0.000000 0.000000
```

The following command changes the amount of memory that vxdmpadm can use to accumulate the statistics:

```
# vxdmpadm iostat start memory=4096
```

The displayed statistics can be filtered by path name, DMP node name, and enclosure name (note that the per-CPU memory has changed following the previous command):

```
# vxdmpadm iostat show pathname=c3t115d0
                     cpu usage = 8132us    per cpu memory = 4096b
                  OPERATIONS            BYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS    WRITES
c3t115d0        0          0          0          0  0.000000 0.000000

# vxdmpadm iostat show dmpnodename=c0t0d0
                     cpu usage = 8501us    per cpu memory = 4096b
                OPERATIONS            BYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS    WRITES
c0t0d0       1088          0     557056          0  0.009542 0.000000

# vxdmpadm iostat show enclosure=Disk
                     cpu usage = 8626us    per cpu memory = 4096b
                OPERATIONS            BYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS    WRITES
c0t0d0       1088          0     557056          0  0.009542 0.000000
```

You can also specify the number of times to display the statistics and the time interval. Here the incremental statistics for a path are displayed twice with a 2-second interval:

```
# vxdmpadm iostat show pathname=c3t115d0 interval=2 count=2
                     cpu usage = 8195us    per cpu memory = 4096b
                OPERATIONS            BYTES            AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS    WRITES
```

```
c3t115d0        0           0           0           0  0.000000 0.000000

                          cpu usage = 59us    per cpu memory = 4096b
                 OPERATIONS              BYTES             AVG TIME(ms)
PATHNAME    READS     WRITES      READS     WRITES     READS    WRITES
c3t115d0        0           0           0           0 0.000000 0.000000
```

Setting the attributes of the paths to an enclosure

You can use the `vxdmpadm setattr` command to set the following attributes of the paths to an enclosure or disk array:

- `active`
 Changes a *standby* (failover) path to an active path. The example below specifies an active path for an A/P-C disk array:

  ```
  # vxdmpadm setattr path c2t10d0 pathtype=active
  ```

- `nomanual`
 Restores the original primary or secondary attributes of a path. This example restores the attributes for a path to an A/P disk array:

  ```
  # vxdmpadm setattr path c3t10d0 pathtype=nomanual
  ```

- `nopreferred`
 Restores the normal priority of a path. The following example restores the default priority to a path:

  ```
  # vxdmpadm setattr path c1t20d0 pathtype=nopreferred
  ```

- `preferred [priority=`*N*`]`
 Specifies a path as preferred, and optionally assigns a priority number to it. If specified, the priority number must be an integer that is greater than or equal to one. Higher priority numbers indicate that a path is able to carry a greater I/O load.

 Note: Setting a priority for path does not change the I/O policy. The I/O policy must be set independently as described in "Specifying the I/O policy" on page 147.

 This example first sets the I/O policy to `priority` for an Active/Active disk array, and then specifies a preferred path with an assigned priority of 2:

  ```
  # vxdmpadm setattr enclosure enc0 iopolicy=priority

  # vxdmpadm setattr path c1t20d0 pathtype=preferred \
      priority=2
  ```

- primary
 Defines a path as being the primary path for an Active/Passive disk array. The following example specifies a primary path for an A/P disk array:

  ```
  # vxdmpadm setattr path c3t10d0 pathtype=primary
  ```

- secondary
 Defines a path as being the secondary path for an Active/Passive disk array. This example specifies a secondary path for an A/P disk array:

  ```
  # vxdmpadm setattr path c4t10d0 pathtype=secondary
  ```

- standby
 Marks a standby (failover) path that it is not used for normal I/O scheduling. This path is used if there are no active paths available for I/O. The next example specifies a standby path for an A/P-C disk array:

  ```
  # vxdmpadm setattr path c2t10d0 pathtype=standby
  ```

Displaying the I/O policy

To display the current and default settings of the I/O policy for an enclosure, array or array type, use the vxdmpadm getattr command.

The following example displays the default and current setting of iopolicy for JBOD disks:

```
# vxdmpadm getattr enclosure Disk iopolicy

ENCLR_NAME        DEFAULT         CURRENT
---------------------------------------
Disk              MinimumQ        Balanced
```

The next example displays the setting of partitionsize for the enclosure enc0, on which the balanced I/O policy with a partition size of 2MB has been set:

```
# vxdmpadm getattr enclosure enc0 partitionsize

ENCLR_NAME        DEFAULT         CURRENT
---------------------------------------
enc0              1024            2048
```

Specifying the I/O policy

You can use the vxdmpadm setattr command to change the I/O policy for distributing I/O load across multiple paths to a disk array or enclosure. You can set policies for an enclosure (for example, HDS01), for all enclosures of a particular type (such as HDS), or for all enclosures of a particular array type (such as A/A for Active/Active, or A/P for Active/Passive).

Note: Starting with release 4.1 of VxVM, I/O policies are recorded in the file `/etc/vx/dmppolicy.info`, and are persistent across reboots of the system. Do not edit this file yourself.

The following policies may be set:

- `adaptive`
 This policy attempts to maximize overall I/O throughput from/to the disks by dynamically scheduling I/O on the paths. It is suggested for use where I/O loads can vary over time. For example, I/O from/to a database may exhibit both long transfers (table scans) and short transfers (random look ups). The policy is also useful for a SAN environment where different paths may have different number of hops. No further configuration is possible as this policy is automatically managed by DMP.
 In this example, the `adaptive` I/O policy is set for the enclosure `enc1`:

  ```
  # vxdmpadm setattr enclosure enc1 iopolicy=adaptive
  ```

- `adaptiveminq`
 Similar to the `adaptive` policy, except that I/O is scheduled according to the length of the I/O queue on each path. The path with the shortest queue is assigned the highest priority.

- `balanced [partitionsize=`*`size`*`]`
 This policy is designed to optimize the use of caching in disk drives and RAID controllers. The size of the cache typically ranges from 120KB to 500KB or more, depending on the characteristics of the particular hardware. During normal operation, the disks (or LUNs) are logically divided into a number of regions (or *partitions*), and I/O from/to a given region is sent on only one of the active paths. Should that path fail, the workload is automatically redistributed across the remaining paths.

You can use the size argument to the `partitionsize` attribute to specify the partition size. The partition size in blocks is adjustable in powers of 2 from 2 up to 2>31 as illustrated in the table below:

Partition size in blocks	Equivalent size in bytes
2	2,048
4	4,096
8	8,192
16	16,384
32	32,768
64	65,536
128	131,072
256	262,144
512	524,288
1024 (default)	1,048,576
2048	2,097,152
4096	4,194,304

The default value for the partition size is 1024 blocks (1MB). A value that is not a power of 2 is silently rounded down to the nearest acceptable value. Specifying a partition size of 0 is equivalent to the default partition size of 1024 blocks (1MB). For example, the suggested partition size for an Hitachi HDS 9960 A/A array is from 16,384 to 65,536 blocks (16MB to 64MB) for an I/O activity pattern that consists mostly of sequential reads or writes.

Note: The benefit of this policy is lost if the value is set larger than the cache size.

The default value can be changed by adjusting the value of a tunable parameter (see "dmp_pathswitch_blks_shift" on page 476).

The next example sets the `balanced` I/O policy with a partition size of 2048 blocks (2MB) on the enclosure `enc0`:

```
# vxdmpadm setattr enclosure enc0 iopolicy=balanced \
partitionsize=2048
```

- `minimumq`
 This policy sends I/O on paths that have the minimum number of outstanding I/O requests in the queue for a LUN. This is suitable for low-end disks or JBODs where a significant track cache does not exist. No further configuration is possible as DMP automatically determines the path with the shortest queue.
 The following example sets the I/O policy to `minimumq` for a JBOD:

  ```
  # vxdmpadm setattr enclosure Disk iopolicy=minimumq
  ```

 This is the default I/O policy for Active/Active (A/A) arrays.

- `priority`
 This policy is useful when the paths in a SAN have unequal performance, and you want to enforce load balancing manually. You can assign priorities to each path based on your knowledge of the configuration and performance characteristics of the available paths, and of other aspects of your system. See "Setting the attributes of the paths to an enclosure" on page 146 for details of how to assign priority values to individual paths.
 In this example, the I/O policy is set to `priority` for all SENA arrays:

  ```
  # vxdmpadm setattr arrayname SENA iopolicy=priority
  ```

- `round-robin`
 This policy shares I/O equally between the paths in a round-robin sequence. For example, if there are three paths, the first I/O request would use one path, the second would use a different path, the third would be sent down the remaining path, the fourth would go down the first path, and so on. No further configuration is possible as this policy is automatically managed by DMP.
 The next example sets the I/O policy to `round-robin` for all Active/Active arrays:

  ```
  # vxdmpadm setattr arraytype A/A iopolicy=round-robin
  ```

 This is the default I/O policy for Active/Passive (A/P) and Asymmetric Active/Active (A/A-A) arrays.

- `singleactive`
 This policy routes I/O down the single active path. This policy can be configured for A/P arrays with one active path per controller, where the other paths are used in case of failover. If configured for A/A arrays, there is no load balancing across the paths, and the alternate paths are only used to provide high availability (HA). If the currently active path fails, I/O is switched to an alternate active path. No further configuration is possible as the single active path is selected by DMP.
 The following example sets the I/O policy to `singleactive` for JBOD disks:

```
# vxdmpadm setattr arrayname DISK iopolicy=singleactive
```

Scheduling I/O on the paths of an Asymmetric Active/Active array

You can specify the `use_all_paths` attribute in conjunction with the `adaptive`, `balanced`, `minimumq`, `priority` and `round-robin` I/O policies to specify whether I/O requests are to be scheduled on the secondary paths in addition to the primary paths of an Asymmetric Active/Active (A/A-A) array. Depending on the characteristics of the array, the consequent improved load balancing can increase the total I/O throughput. However, this feature should only be enabled if recommended by the array vendor. It has no effect for array types other than A/A-A.

For example, the following command sets the `balanced` I/O policy with a partition size of 2048 blocks (2MB) on the enclosure `enc0`, and allows scheduling of I/O requests on the secondary paths:

```
# vxdmpadm setattr enclosure enc0 iopolicy=balanced \
  partitionsize=2048 use_all_paths=no
```

The default setting for this attribute is `use_all_paths=no`.

Example of applying load balancing in a SAN

This example describes how to configure load balancing in a SAN environment where there are multiple primary paths to an Active/Passive device through several SAN switches. As can be seen in this sample output from the `vxdisk list` command, the device `c3t2d15` has eight primary paths:

```
# vxdisk list c3t2d15

Device: c3t2d15
...
numpaths: 8
c2t0d15 state=enabled type=primary
c2t1d15 state=enabled type=primary
c3t1d15 state=enabled type=primary
c3t2d15 state=enabled type=primary
c4t2d15 state=enabled type=primary
c4t3d15 state=enabled type=primary
c5t3d15 state=enabled type=primary
c5t4d15 state=enabled type=primary
```

In addition, the device is in the enclosure `ENC0`, belongs to the disk group `mydg`, and contains a simple concatenated volume `myvol1`.

The first step is to enable the gathering of DMP statistics:

```
# vxdmpadm iostat start
```

Next the `dd` command is used to apply an input workload from the volume:

```
# dd if=/dev/vx/rdsk/mydg/myvol1 of=/dev/null &
```

By running the `vxdmpadm iostat` command to display the DMP statistics for the device, it can be seen that all I/O is being directed to one path, `c5t4d15`:

```
# vxdmpadm iostat show dmpnodename=c3t2d15 interval=5 count=2
...
cpu usage = 11294us per cpu memory = 32768b
             OPERATIONS           KBYTES           AVG TIME(ms)
PATHNAME    READS   WRITES    READS     WRITES    READS       WRITES
c2t0d15     0       0         0         0         0.000000    0.000000
c2t1d15     0       0         0         0         0.000000    0.000000
c3t1d15     0       0         0         0         0.000000    0.000000
c3t2d15     0       0         0         0         0.000000    0.000000
c4t2d15     0       0         0         0         0.000000    0.000000
c4t3d15     0       0         0         0         0.000000    0.000000
c5t3d15     0       0         0         0         0.000000    0.000000
c5t4d15     5493    0         5493      0         0.411069    0.000000
```

The `vxdmpadm` command is used to display the I/O policy for the enclosure that contains the device:

```
# vxdmpadm getattr enclosure ENC0 iopolicy

ENCLR_NAME      DEFAULT           CURRENT
============================================
ENC0            Round-Robin       Single-Active
```

This shows that the policy for the enclosure is set to `singleactive`, which explains why all the I/O is taking place on one path.

To balance the I/O load across the multiple primary paths, the policy is set to `round-robin` as shown here:

```
# vxdmpadm setattr enclosure ENC0 iopolicy=round-robin
# vxdmpadm getattr enclosure ENC0 iopolicy

ENCLR_NAME      DEFAULT           CURRENT
============================================
ENC0            Round-Robin       Round-Robin
```

The DMP statistics are now reset:

```
# vxdmpadm iostat reset
```

With the workload still running, the effect of changing the I/O policy to balance the load across the primary paths can now be seen.

```
# vxdmpadm iostat show dmpnodename=c3t2d15 interval=5 \
     count=2
...
cpu usage = 14403us per cpu memory = 32768b
             OPERATIONS           KBYTES           AVG TIME(ms)
PATHNAME    READS   WRITES    READS     WRITES    READS       WRITES
c2t0d15     1021    0         1021      0         0.396670    0.000000
c2t1d15     947     0         947       0         0.391763    0.000000
c3t1d15     1004    0         1004      0         0.393426    0.000000
c3t2d15     1027    0         1027      0         0.402142    0.000000
```

```
c4t2d15     1086    0         1086      0         0.390424    0.000000
c4t3d15     1048    0         1048      0         0.391221    0.000000
c5t3d15     1036    0         1036      0         0.390927    0.000000
c5t4d15     1021    0         1021      0         0.392752    0.000000
```

The enclosure can be returned to the single active I/O policy by entering the following command:

```
# vxdmpadm setattr enclosure ENC0 iopolicy=singleactive
```

Disabling I/O for paths, controllers or array ports

Note: From release 5.0 of VxVM, this operation is supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

Disabling I/O through a path, HBA controller or array port prevents DMP from issuing I/O requests through the specified path, or the paths that are connected to the specified controller or array port. The command blocks until all pending I/O requests issued through the paths are completed.

To disable I/O for a path, use the following command:

```
# vxdmpadm [-c|-f] disable path=path_name
```

To disable I/O for the paths connected to an HBA controller, use the following command:

```
# vxdmpadm [-c|-f] disable ctlr=ctlr_name
```

To disable I/O for the paths connected to an array port, use one of the following commands:

```
# vxdmpadm [-c|-f] disable enclosure=enclr_name \
  portid=array_port_ID
# vxdmpadm [-c|-f] disable pwwn=array_port_WWN
```

where the array port is specified either by the enclosure name and the array port ID, or by the array port's worldwide name (WWN) identifier.

The following are examples of using the command to disable I/O on an array port:

```
# vxdmpadm disable enclosure=HDS9500V0 portid=1A
# vxdmpadm disable pwwn=20:00:00:E0:8B:06:5F:19
```

You can use the -c option to check if there is only a single active path to the disk. If so, the `disable` command fails with an error message unless you use the `-f` option to forcibly disable the path.

The `disable` operation fails if it is issued to a controller that is connected to the root disk through a single path, and there are no root disk mirrors configured on alternate paths. If such mirrors exist, the command succeeds.

Enabling I/O for paths, controllers or array ports

Note: This operation is not supported for controllers that are used to access disk arrays on which cluster-shareable disk groups are configured.

Enabling a controller allows a previously disabled path, HBA controller or array port to accept I/O again. This operation succeeds only if the path, controller or array port is accessible to the host, and I/O can be performed on it. When connecting Active/Passive disk arrays, the `enable` operation results in failback of I/O to the primary path. The `enable` operation can also be used to allow I/O to the controllers on a system board that was previously detached.

To enable I/O for a path, use the following command:

```
# vxdmpadm enable path=path_name
```

To enable I/O for the paths connected to an HBA controller, use the following command:

```
# vxdmpadm enable ctlr=ctlr_name
```

To enable I/O for the paths connected to an array port, use one of the following commands:

```
# vxdmpadm enable enclosure=enclr_name portid=array_port_ID
# vxdmpadm [-f] disable pwwn=array_port_WWN
```

where the array port is specified either by the enclosure name and the array port ID, or by the array port's worldwide name (WWN) identifier.

The following are examples of using the command to enable I/O on an array port:

```
# vxdmpadm enable enclosure=HDS9500V0 portid=1A
# vxdmpadm enable pwwn=20:00:00:E0:8B:06:5F:19
```

Upgrading disk controller firmware

You can upgrade disk controller firmware without performing a system reboot or unloading the VxVM drivers.

First obtain the appropriate firmware upgrades from your disk drive vendor. You can usually download the appropriate files and documentation from the vendor's support website.

For a system with a volume mirrored across 2 controllers on one HBA, set up the configuration as follows:

1 Disable the plex that is associated with the disk device:

```
# /opt/VRTS/bin/vxplex -g diskgroup det plex
```

2 Stop I/O to all disks through one controller of the HBA:

```
# /opt/VRTS/bin/vxdmpadm disable ctlr=first_cntlr
```

For the other controller on the HBA, enter:

```
# /opt/VRTS/bin/vxdmpadm -f disable ctlr=second_cntlr
```

3 Upgrade the firmware on those disks for which the controllers have been disabled using the procedures that you obtained from the disk drive vendor.

4 After doing the upgrade, re-enable all the controllers:

```
# /opt/VRTS/bin/vxdmpadm enable ctlr=first_cntlr
# /opt/VRTS/bin/vxdmpadm enable ctlr=second_cntlr
```

5 Re-enable the plex associated with the device:

```
# /opt/VRTS/bin/vxplex -g diskgroup att volume plex
```

This command takes some time depending upon the size of the mirror set.

Renaming an enclosure

The `vxdmpadm setattr` command can be used to assign a meaningful name to an existing enclosure, for example:

```
# vxdmpadm setattr enclosure enc0 name=GRP1
```

This example changes the name of an enclosure from enc`0` to `GRP1`.

Note: The maximum length of the enclosure name prefix is 25 characters. The name must not contain an underbar character (_).

The following command shows the changed name:

```
# vxdmpadm listenclosure all

ENCLR_NAME   ENCLR_TYPE     ENCLR_SNO                  STATUS
============================================================
other0       OTHER          OTHER_DISKS                CONNECTED
jbod0        X1             X1_DISKS                   CONNECTED
GRP1         ACME           60020f20000001a90000       CONNECTED
```

Configuring the response to I/O failures

By default, DMP is configured to retry a failed I/O request up to 5 times for a single path. To display the current settings for handling I/O request failures that are applied to the paths to an enclosure, array name or array type, use the `vxdmpadm getattr` command:

```
# vxdmpadm getattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption
```

See "Displaying recoveryoption values" on page 159 for more information.

The following example displays the I/O request failure setting for the paths to the enclosure `enc0`:

```
# vxdmpadm getattr enclosure enc0 recoveryoption
```

The `vxdmpadm setattr` command can be used to configure how DMP responds to failed I/O requests on the paths to a specified enclosure, disk array name, or type of array.

The following form of the command sets a limit for the number of times that DMP will attempt to retry sending an I/O request on a path:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=fixedretry retrycount==n
```

The value of the argument to `retrycount` specifies the number of retries to be attempted before DMP reschedules the I/O request on another available path, or fails the request altogether.

As an alternative to specifying a fixed number of retries, the following version of the command specifies how long DMP should allow an I/O request to be retried on a path:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type} \
  recoveryoption=timebound iotimeout==seconds
```

The value of the argument to `iotimeout` specifies the time in seconds that DMP waits for an outstanding I/O request to succeed before it reschedules the request on another available path, or fails the I/O request altogether. The effective number of retries is the value of `iotimeout` divided by the sum of the times taken for each retry attempt. DMP abandons retrying to send the I/O request before the specified time limit has expired if it predicts that the next retry will take the total elapsed time over this limit.

The default value of `iotimeout` is 10 seconds. For some applications, such as Oracle, it may be desirable to set `iotimeout` to a larger value, such as 60 seconds.

Note: The `fixedretry` and `timebound` settings are mutually exclusive.

The following example configures time-bound recovery for the enclosure `enc0`, and sets the value of `iotimeout` to 60 seconds:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=timebound \
  iotimeout=60
```

The next example sets a fixed-retry limit of 10 for the paths to all Active/Active arrays:

```
# vxdmpadm setattr arraytype A/A recoveryoption=fixedretry \
  retrycount=10
```

Specifying `recoveryoption=default` resets DMP to the default settings corresponding to `recoveryoption=fixedretry retrycount=30`, for example:

```
# vxdmpadm setattr arraytype A/A recoveryoption=default
```

This command also has the effect of configuring I/O throttling with an I/O timeout value of 10 seconds on the paths.

See "Configuring the I/O throttling mechanism" on page 157 for details.

Note: The response to I/O failure settings is persistent across reboots of the system.

Configuring the I/O throttling mechanism

By default, II/O throttling is turned on for all paths with DMP configured to wait for a maximum of 10 seconds for outstanding I/O requests to succeed. To display the current settings for I/O throttling that are applied to the paths to an enclosure, array name or array type, use the `vxdmpadm getattr` command:

```
# vxdmpadm getattr \
  {enclosure enc-name|arrayname name|arraytype type}\
  recoveryoption
```

See "Displaying recoveryoption values" on page 159 for more information.

The following example displays the I/O throttling setting for the paths to the enclosure `enc0`:

```
# vxdmpadm getattr enclosure enc0 recoveryoption
```

If enabled, I/O throttling imposes a small overhead on CPU and memory usage because of the activity of the statistics-gathering daemon. If I/O throttling is disabled, the daemon no longer collects statistics, and remains inactive until I/O throttling is re-enabled.

To turn off I/O throttling, use the following form of the `vxdmpadm setattr` command:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type}\
  recoveryoption=nothrottle
```

The following example shows how to disable I/O throttling for the paths to the enclosure enc0:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=nothrottle
```

The vxdmpadm setattr command can be used to enable I/O throttling on the paths to a specified enclosure, disk array name, or type of array:

```
# vxdmpadm setattr \
  {enclosure enc-name|arrayname name|arraytype type}\
  recoveryoption=throttle {iotimeout=seconds|queuedepth=n}
```

If the iotimeout attribute is specified, its argument specifies the time in seconds that DMP waits for an outstanding I/O request to succeed before invoking I/O throttling on the path. The default value of iotimeout is 10 seconds. Setting iotimeout to a larger value potentially causes more I/O requests to become queued up in the SCSI driver before I/O throttling is invoked.

If the queuedepth attribute is specified, its argument specifies the number of I/O requests that can be outstanding on a path before DMP invokes I/O throttling. The default value of queuedepth is 40. Setting queuedepth to a larger value allows more I/O requests to become queued up in the SCSI driver before I/O throttling is invoked.

Note: The iotimeout and queuedepth attributes are mutually exclusive.

The following example sets the value of iotimeout to 60 seconds for the enclosure enc0:

```
# vxdmpadm setattr enclosure enc0 recoveryoption=throttle \
  iotimeout=60
```

The next example sets the value of queuedepth to 30 for the paths to all Active/Active arrays:

```
# vxdmpadm setattr arraytype A/A recoveryoption=throttle \
  queuedepth=30
```

Specifying recoveryoption=default resets I/O throttling to the default settings corresponding to recoveryoption=throttle iotimeout=10, for example:

```
# vxdmpadm setattr arraytype A/A recoveryoption=default
```

This command also has the effect of configuring a fixed-retry limit of 30 on the paths.

See "Configuring the response to I/O failures" on page 156 for details.

Note: The I/O throttling settings are persistent across reboots of the system.

Displaying recoveryoption values

The following example shows the `vxdmpadm getattr command` being used to display the `recoveryoption` option values that are set on an enclosure.

```
# vxdmpadm getattr enclosure HDS9500-ALUA0 recoveryoption
ENCLR-NAME     RECOVERY-OPTION  DEFAULT[VAL]    CURRENT[VAL]
============================================================
HDS9500-ALUA0  Throttle         Timebound[10]  Queuedepth[60]
HDS9500-ALUA0  Error-Retry      Fixed-Retry[30]Timebound[20]
```

This shows the default and current policy options and their values. The possible option settings are summarized in the following table.

Recovery option	Possible settings	Description
Error-Retry settings:		
recoveryoption=fixedretry	Fixed-Retry (retrycount)	DMP retries a failed I/O request for the specified number of times if I/O fails.
recoveryoption=timebound	Timebound (iotimeout)	DMP retries a failed I/O request after the specified time in seconds if I/O fails.
Throttle settings:		
recoveryoption=nothrottle	None	Not applicable
recoveryoption=throttle	Queuedepth (queuedepth)	DMP throttles the path if the specified number of queued I/O requests is exceeded.
	Timebound (iotimeout)	DMP throttles the path if an I/O request does not return within the specified time in seconds.

Configuring DMP path restoration policies

DMP maintains a kernel thread that re-examines the condition of paths at a specified interval. The type of analysis that is performed on the paths depends on the checking policy that is configured.

Note: The DMP path restoration thread does not change the disabled state of the path through a controller that you have disabled using `vxdmpadm disable`.

Use the `start restore` command to configure one of the following policies:

- `check_all`

 The path restoration thread analyzes all paths in the system and revives the paths that are back online, as well as disabling the paths that are inaccessible. The command to configure this policy is:

  ```
  # vxdmpadm start restore [interval=seconds] policy=check_all
  ```

- `check_alternate`

 The path restoration thread checks that at least one alternate path is healthy. It generates a notification if this condition is not met. This policy avoids inquiry commands on all healthy paths, and is less costly than `check_all` in cases where a large number of paths are available. This policy is the same as `check_all` if there are only two paths per DMP node. The command to configure this policy is:

  ```
  # vxdmpadm start restore [interval=seconds] \
    policy=check_alternate
  ```

- `check_disabled`

 This is the default path restoration policy. The path restoration thread checks the condition of paths that were previously disabled due to hardware failures, and revives them if they are back online. The command to configure this policy is:

  ```
  # vxdmpadm start restore [interval=seconds] \
    policy=check_disabled
  ```

- `check_periodic`

 The path restoration thread performs `check_all` once in a given number of cycles, and `check_disabled` in the remainder of the cycles. This policy may lead to periodic slowing down (due to `check_all`) if there is a large number of paths available. The command to configure this policy is:

  ```
  # vxdmpadm start restore interval=seconds \
    policy=check_periodic [period=number]
  ```

The `interval` attribute must be specified for this policy. The default number of cycles between running the `check_all` policy is 10.

The interval attribute specifies how often the path restoration thread examines the paths. For example, after stopping the path restoration thread, the polling interval can be set to 400 seconds using the following command:

```
# vxdmpadm start restore interval=400
```

Note: The default interval is 300 seconds. Decreasing this interval can adversely affect system performance.

To change the interval or policy, first stop the path restoration thread as described in "Stopping the DMP path restoration thread" on page 161, and then restart it with new attributes.

See the `vxdmpadm`(1M) manual page for more information about DMP restore policies.

Stopping the DMP path restoration thread

Use the following command to stop the DMP path restoration thread:

```
# vxdmpadm stop restore
```

Note: Automatic path failback stops if the path restoration thread is stopped.

Displaying the status of the DMP path restoration thread

Use the following command to display the status of the automatic path restoration kernel thread, its polling interval, and the policy that it uses to check the condition of paths:

```
# vxdmpadm stat restored
```

This produces output such as the following:

```
The number of daemons running : 1
The interval of daemon: 300
The policy of daemon: check_disabled
```

Displaying information about the DMP error-handling thread

To display information about the kernel thread that handles DMP errors, use the following command:

```
# vxdmpadm stat errord
```

One daemon should be shown as running.

Configuring array policy modules

An array policy module (APM) is a dynamically loadable kernel module that may be provided by some vendors for use in conjunction with an array. An APM defines procedures to:

- Select an I/O path when multiple paths to a disk within the array are available.
- Select the path failover mechanism.
- Select the alternate path in the case of a path failure.
- Put a path change into effect.
- Respond to SCSI reservation or release requests.

DMP supplies default procedures for these functions when an array is registered. An APM may modify some or all of the existing procedures that are provided by DMP or by another version of the APM.

You can use the following command to display all the APMs that are configured for a system:

```
# vxdmpadm listapm all
```

The output from this command includes the file name of each module, the supported array type, the APM name, the APM version, and whether the module is currently in use (*loaded*). To see detailed information for an individual module, specify the module name as the argument to the command:

```
# vxdmpadm listapm module_name
```

To add and configure an APM, use the following command:

```
# vxdmpadm -a cfgapm module_name [attr1=value1 \
  [attr2=value2 ...]]
```

The optional configuration attributes and their values are specific to the APM for an array. Consult the documentation that is provided by the array vendor for details.

Note: By default, DMP uses the most recent APM that is available. Specify the `-u` option instead of the `-a` option if you want to force DMP to use an earlier version of the APM. The current version of an APM is replaced only if it is not in use.

Specifying the `-r` option allows you to remove an APM that is not currently loaded:

```
# vxdmpadm -r cfgapm module_name
```

For more information about configuring APMs, see the `vxdmpadm`(1M) manual page.

Chapter 4

Creating and administering disk groups

This chapter describes how to create and manage *disk groups*. Disk groups are named collections of disks that share a common configuration. Volumes are created within a disk group and are restricted to using disks within that disk group.

Note: In releases of Veritas Volume Manager (VxVM) prior to 4.0, a system installed with VxVM was configured with a default disk group, `rootdg`, that had to contain at least one disk. By default, operations were directed to the `rootdg` disk group. From release 4.0 onward, VxVM can function without any disk group having been configured. Only when the first disk is placed under VxVM control must a disk group be configured. There is no longer a requirement that you name any disk group `rootdg`, and any disk group that is named `rootdg` has no special properties because of this name. See "Specifying a disk group to commands" on page 167 for more information about using disk group names that are reserved for special purposes.

Additionally, prior to VxVM 4.0, some commands such as `vxdisk` were able to deduce the disk group if the name of an object was uniquely defined in one disk group among all the imported disk groups. Resolution of a disk group in this way is no longer supported for any command.

For a discussion of disk groups that are compatible with the Cross-platform Data Sharing (CDS) feature of Veritas Volume Manager, see the *Veritas Storage Foundation Cross-Platform Data Sharing Administrator's Guide*. The CDS feature allows you to move VxVM disks and objects between machines that are running under different operating systems.

As system administrator, you can create additional disk groups to arrange your system's disks for different purposes. Many systems do not use more than one disk group, unless they have a large number of disks. Disks can be initialized, reserved, and added to disk groups at any time. Disks need not be added to disk groups until the disks are needed to create VxVM objects.

When a disk is added to a disk group, it is given a name (for example, `mydg02`). This name identifies a disk for operations such as volume creation or mirroring. The name also relates directly to the underlying physical disk. If a physical disk is moved to a different target address or to a different controller, the name `mydg02` continues to refer to it. Disks can be replaced by first associating a different physical disk with the name of the disk to be replaced and then recovering any volume data that was stored on the original disk (from mirrors or backup copies).

Having disk groups that contain many disks and VxVM objects causes the private region to fill. In the case of large disk groups that are expected to contain more than several hundred disks and VxVM objects, disks should be set up with larger private areas. A major portion of a private region provides space for a disk group configuration database that contains records for each VxVM object in that disk group. Because each configuration record takes up approximately 256 bytes, the number of records that can be created in a disk group can be estimated from the configuration database copy size. The copy size in blocks can be obtained from the output of the command `vxdg list` *diskgroup* as the value of the `permlen` parameter on the line starting with the string "`config:`". This value is the smallest of the `len` values for all copies of the configuration database in the disk group. The amount of remaining free space in the configuration database is shown as the value of the `free` parameter. An example is shown in "Displaying disk group information" on page 169. One way to overcome the problem of running out of free space is to split the affected disk group into two separate disk groups. See "Reorganizing the contents of disk groups" on page 195 for details.

For information on backing up and restoring disk group configurations, see "Backing up and restoring disk group configuration data" on page 213.

Specifying a disk group to commands

Note: Most VxVM commands require superuser or equivalent privileges.

Many VxVM commands allow you to specify a disk group using the `-g` option. For example, the following command creates a volume in the disk group, `mktdg`:

```
# vxassist -g mktdg make mktvol 5g
```

The block special device corresponding to this volume is:

```
/dev/vx/dsk/mktdg/mktvol
```

System-wide reserved disk groups

The following disk group names are reserved, and cannot be used to name any disk groups that you create:

`bootdg` Specifes the boot disk group. This is an alias for the disk group that contains the volumes that are used to boot the system. VxVM sets `bootdg` to the appropriate disk group if it takes control of the root disk. Otherwise, `bootdg` is set to `nodg` (no disk group; see below).

Caution: Do not attempt to change the assigned value of `bootdg`. Doing so may render your system unbootable.

`defaultdg` Specifies the default disk group. This is an alias for the disk group name that should be assumed if the `-g` option is not specified to a command, or if the `VXVM_DEFAULTDG` environment variable is undefined. By default, `defaultdg` is set to `nodg` (no disk group; see below).

`nodg` Specifies to an operation that no disk group has been defined. For example, if the root disk is not under VxVM control, `bootdg` is set to `nodg`.

Note: If you have upgraded your system, you may find it convenient to continue to configure a disk group named `rootdg` as the default disk group (`defaultdg`). There is no requirement that both `defaultdg` and `bootdg` refer to the same disk group, nor that either the default disk group or the boot disk group be named `rootdg`.

Rules for determining the default disk group

It is recommended that you use the -g option to specify a disk group to VxVM commands that accept this option. If you do not specify the disk group, VxVM applies rules in the following order until it determines a disk group name:

- Use the default disk group name that is specified by the environment variable VXVM_DEFAULTDG. This variable can also be set to one of the reserved system-wide disk group names: bootdg, defaultdg, or nodg. If the variable is undefined, the following rule is applied.
- Use the disk group that has been assigned to the system-wide default disk group alias, defaultdg. See "Displaying and specifying the system-wide default disk group" on page 168. If this alias is undefined, the following rule is applied.
- If the operation can be performed without requiring a disk group name (for example, an edit operation on disk access records), do so.

If none of these rules succeeds, the requested operation fails.

Caution: In releases of VxVM prior to 4.0, a subset of commands attempted to deduce the disk group by searching for the object name that was being operated upon by a command. This functionality is no longer supported. Scripts that rely on deducing the disk group from an object name may fail.

Displaying the system-wide boot disk group

To display the currently defined system-wide boot disk group, use the following command:

```
# vxdg bootdg
```

See the vxdg(1M) manual page for more information.

Displaying and specifying the system-wide default disk group

To display the currently defined system-wide default disk group, use the following command:

```
# vxdg defaultdg
```

If a default disk group has not been defined, nodg is displayed. Alternatively, you can use the following command to display the default disk group:

```
# vxprint -Gng defaultdg 2>/dev/null
```

In this case, if there is no default disk group, nothing is displayed.

Use the following command to specify the name of the disk group that is aliased by defaultdg:

```
# vxdctl defaultdg diskgroup
```

If bootdg is specified as the argument to this command, the default disk group is set to be the same as the currently defined system-wide boot disk group.

If nodg is specified as the argument to the vxdctl defaultdg command, the default disk group is undefined.

Note: The specified *diskgroup* need not currently exist on the system.

See the vxdctl(1M) and vxdg(1M) manual pages for more information.

Displaying disk group information

To display information on existing disk groups, enter the following command:

```
# vxdg list
NAME    STATE     ID
rootdg  enabled   730344554.1025.tweety
newdg   enabled   731118794.1213.tweety
```

To display more detailed information on a specific disk group, use the following command:

```
# vxdg list diskgroup
```

The output from this command when applied to a disk group named mydg is similar to the following:

```
# vxdg list mydg
Group: mydg
dgid: 962910960.1025.bass
import-id: 0.1
flags:
version: 140
local-activation: read-write
alignment : 512 (bytes)
ssb: on
detach-policy: local
copies: nconfig=default nlog=default
config: seqno=0.1183 permlen=3448 free=3428 templen=12
loglen=522
config disk c0t10d0 copy 1 len=3448 state=clean online
config disk c0t11d0 copy 1 len=3448 state=clean online
log disk c0t10d0 copy 1 len=522
log disk c0t11d0 copy 1 len=522
```

To verify the disk group ID and name associated with a specific disk (for example, to import the disk group), use the following command:

```
# vxdisk -s list devicename
```

This command provides output that includes the following information for the specified disk. For example, output for disk c0t12d0 as follows:

```
Disk: c0t12d0
type:   simple
```

```
flags:  online ready private autoconfig autoimport imported
diskid: 963504891.1070.bass
dgname: newdg
dgid:   963504895.1075.bass
hostid: bass
info:   privoffset=128
```

Displaying free space in a disk group

Before you add volumes and file systems to your system, make sure you have enough free disk space to meet your needs.

To display free space in the system, use the following command:

```
# vxdg free
```

The following is example output:

```
GROUP DISK        DEVICE      TAG         OFFSET    LENGTH    FLAGS
mydg  mydg01      c0t10d0     c0t10d0     0         4444228   -
mydg  mydg02      c0t11d0     c0t11d0     0         4443310   -
newdg newdg01     c0t12d0     c0t12d0     0         4443310   -
newdg newdg02     c0t13d0     c0t13d0     0         4443310   -
oradg oradg01     c0t14d0     c0t14d0     0         4443310   -
```

To display free space for a disk group, use the following command:

```
# vxdg -g diskgroup free
```

where `-g` *diskgroup* optionally specifies a disk group.

For example, to display the free space in the disk group, `mydg`, use the following command:

```
# vxdg -g mydg free
```

The following example output shows the amount of free space in sectors:

```
DISK    DEVICE      TAG         OFFSET    LENGTH      FLAGS
mydg01  c0t10d0     c0t10d0     0         4444228     -
mydg02  c0t11d0     c0t11d0     0         4443310     -
```

Creating a disk group

Data related to a particular set of applications or a particular group of users may need to be made accessible on another system. Examples of this are:

- A system has failed and its data needs to be moved to other systems.
- The work load must be balanced across a number of systems.

Disks must be placed in one or more disk groups before VxVM can use the disks for volumes. It is important that you locate data related to particular applications or users on an identifiable set of disks. When you need to move these disks, this allows you to move only the application or user data that should be moved.

A disk group must have at least one disk associated with it. A new disk group can be created when you use menu item `1 (Add or initialize one or more disks)` of the `vxdiskadm` command to add disks to VxVM control, as described in "Adding a disk to VxVM" on page 97. The disks to be added to a disk group must not belong to an existing disk group.

You can also use the `vxdiskadd` command to create a new disk group:

```
# vxdiskadd c1t0d0
```

where `c1t0d0` in this example is the device name of a disk that is not currently assigned to a disk group. The command dialog is similar to that described for the `vxdiskadm` command in "Adding a disk to VxVM" on page 97.

Disk groups can also be created by using the `vxdg init` command:

```
# vxdg init diskgroup [cds=on|off] diskname=devicename
```

For example, to create a disk group named `mktdg` on device `c1t0d0`:

```
# vxdg init mktdg mktdg01=c1t0d0
```

The disk specified by the device name, `c1t0d0`, must have been previously initialized with `vxdiskadd` or `vxdiskadm`, and must not currently belong to a disk group.

You can use the `cds` attribute with the `vxdg init` command to specify whether a new disk group is compatible with the Cross-platform Data Sharing (CDS) feature. In Veritas Volume Manager 4.0 and later releases, newly created disk groups are compatible with CDS by default (equivalent to specifying `cds=on`). If you want to change this behavior, edit the file `/etc/default/vxdg`, and set the attribute-value pair `cds=off` in this file before creating a new disk group.

Alternatively, you can use the following command to set this attribute for a disk group:

```
# vxdg -g diskgroup set cds=on|off
```

Adding a disk to a disk group

To add a disk to an existing disk group, use menu item `1 (Add or initialize one or more disks)` of the `vxdiskadm` command. For details of this procedure, see "Adding a disk to VxVM" on page 97.

You can also use the `vxdiskadd` command to add a disk to a disk group, for example:

```
# vxdiskadd c1t1d0
```

where `c1t1d0` is the device name of a disk that is not currently assigned to a disk group. The command dialog is similar to that described for the `vxdiskadm` command in "Adding a disk to VxVM" on page 97.

Removing a disk from a disk group

Note: Before you can remove the last disk from a disk group, you must disable the disk group as described in "Disabling a disk group" on page 207. Alternatively, you can destroy the disk group as described in "Destroying a disk group" on page 208.

A disk that contains no subdisks can be removed from its disk group with this command:

```
# vxdg [-g diskgroup] rmdisk diskname
```

For example, to remove `mydg02` from the disk group, `mydg`, use this command:

```
# vxdg -g mydg rmdisk mydg02
```

If the disk has subdisks on it when you try to remove it, the following error message is displayed:

```
VxVM vxdg ERROR V-5-1-552 Disk diskname is used by one or more
subdisks
Use -k to remove device assignment.
```

Using the `-k` option allows you to remove the disk even if subdisks are present. For more information, see the `vxdg`(1M) manual page.

Caution: Use of the `-k` option to `vxdg` can result in data loss.

Once the disk has been removed from its disk group, you can (optionally) remove it from VxVM control completely, as follows:

```
# vxdiskunsetup devicename
```

For example, to remove `c1t0d0` from VxVM control, use these commands:

```
# vxdiskunsetup c1t0d0
```

You can remove a disk on which some subdisks of volumes are defined. For example, you can consolidate all the volumes onto one disk. If you use `vxdiskadm` to remove a disk, you can choose to move volumes off that disk. To do this, run `vxdiskadm` and select item `2` (`Remove a disk`) from the main menu.

If the disk is used by some volumes, this message is displayed:

```
VxVM ERROR V-5-2-369 The following volumes currently use part of
disk mydg02:

    home usrvol

    Volumes must be moved from mydg02 before it can be removed.

    Move volumes to other disks? [y,n,q,?] (default: n)
```

If you choose **y**, then all volumes are moved off the disk, if possible. Some volumes may not be movable. The most common reasons why a volume may not be movable are as follows:

- There is not enough space on the remaining disks.
- Plexes or striped subdisks cannot be allocated on different disks from existing plexes or striped subdisks in the volume.

If `vxdiskadm` cannot move some volumes, you may need to remove some plexes from some disks to free more space before proceeding with the disk removal operation.

Deporting a disk group

Deporting a disk group disables access to a disk group that is currently enabled (imported) by the system. Deport a disk group if you intend to move the disks in a disk group to another system. Also, deport a disk group if you want to use all of the disks remaining in a disk group for a new purpose.

To deport a disk group

1. Stop all activity by applications to volumes that are configured in the disk group that is to be deported. Unmount file systems and shut down databases that are configured on the volumes.

 Note: Deportation fails if the disk group contains volumes that are in use (for example, by mounted file systems or databases).

2. Use the following command to stop the volumes in the disk group:

   ```
   # vxvol -g diskgroup stopall
   ```

3. Select menu item `8 (Remove access to (deport) a disk group)` from the `vxdiskadm` main menu.

4. At the following prompt, enter the name of the disk group to be deported (in this example, `newdg`):

   ```
   Remove access to (deport) a disk group
   Menu: VolumeManager/Disk/DeportDiskGroup

   Use this menu operation to remove access to
   a disk group that is currently enabled (imported) by this
   system.
   Deport a disk group if you intend to move the disks in a disk
   group to another system. Also, deport a disk group if you
   want to use all of the disks remaining in a disk group for some
   new purpose.

   You will be prompted for the name of a disk group. You will
   also be asked if the disks should be disabled (offlined). For
   removable disk devices on some systems, it is important to
   disable all access to the disk before removing the disk.
   ```

```
Enter name of disk group [<group>,list,q,?] (default: list)
newdg
```

5 At the following prompt, enter `y` if you intend to remove the disks in this disk group:

```
VxVM INFO V-5-2-377 The requested operation is to disable
access to the removable disk group named newdg. This disk
group is stored on the following disks:
   newdg01 on device c1t1d0

You can choose to disable access to (also known as "offline")
these disks. This may be necessary to prevent errors if you
actually remove any of the disks from the system.

Disable (offline) the indicated disks? [y,n,q,?]
(default: n) y
```

6 At the following prompt, press Return to continue with the operation:

```
Continue with operation? [y,n,q,?] (default: y)
```

Once the disk group is deported, the `vxdiskadm` utility displays the following message:

```
VxVM INFO V-5-2-269 Removal of disk group newdg was
successful.
```

7 At the following prompt, indicate whether you want to disable another disk group (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Disable another disk group? [y,n,q,?] (default: n)
```

Alternatively, you can use the `vxdg` command to deport a disk group:

```
# vxdg deport diskgroup
```

Importing a disk group

Importing a disk group enables access by the system to a disk group. To move a disk group from one system to another, first disable (deport) the disk group on the original system, and then move the disk between systems and enable (import) the disk group.

To import a disk group

1 Use the following command to ensure that the disks in the deported disk group are online:

```
# vxdisk -s list
```

2 Select menu item `7 (Enable access to (import) a disk group)` from the `vxdiskadm` main menu.

3 At the following prompt, enter the name of the disk group to import (in this example, `newdg`):

```
Enable access to (import) a disk group
Menu: VolumeManager/Disk/EnableDiskGroup

Use this operation to enable access to a
disk group. This can be used as the final part of moving a disk
group from one system to another. The first part of moving a
disk group is to use the "Remove access to (deport) a disk
group" operation on the original host.

A disk group can be imported from another host that failed
without first deporting the disk group. Be sure that all disks
in the disk group are moved between hosts.

If two hosts share a SCSI bus, be very careful to ensure that
the other host really has failed or has deported the disk
group.
If two active hosts import a disk group at the same time, the
disk group will be corrupted and will become unusable.

Select disk group to import [<group>,list,q,?] (default: list)
newdg
```

Once the import is complete, the `vxdiskadm` utility displays the following success message:

```
VxVM INFO V-5-2-374 The import of newdg was successful.
```

4 At the following prompt, indicate whether you want to import another disk group (`y`) or return to the `vxdiskadm` main menu (`n`):

```
Select another disk group? [y,n,q,?] (default: n)
```

Alternatively, you can use the `vxdg` command to import a disk group:

```
# vxdg import diskgroup
```

Handling disks with duplicated identifiers

Advanced disk arrays provide hardware tools that you can use to create clones of existing disks outside the control of VxVM. For example, these disks may have been created as hardware snapshots or mirrors of existing disks in a disk group. As a result, the VxVM private region is also duplicated on the cloned disk. When the disk group containing the original disk is subsequently imported, VxVM detects multiple disks that have the same disk identifier that is defined in the private region. In releases prior to 5.0, if VxVM could not determine which disk was the original, it would not import such disks into the disk group. The duplicated disks would have to be re-initialized before they could be imported.

From release 5.0, a unique disk identifier (UDID) is added to the disk's private region when the disk is initialized or when the disk is imported into a disk group (if this identifier does not already exist). Whenever a disk is brought online, the current UDID value that is known to the Device Discovery Layer (DDL) is

compared with the UDID that is set in the disk's private region. If the UDID values do not match, the udid_mismatch flag is set on the disk. This flag can be viewed with the vxdisk list command.

A new set of vxdisk and vxdg operations are provided to handle such disks; either by writing the DDL value of the UDID to a disk's private region, or by tagging a disk and specifying that it is a cloned disk to the vxdg import operation.

The following is sample output from the vxdisk list command showing that disks c2t66d0, c2t67d0 and c2t68d0 are marked with the udid_mismatch flag:

```
# vxdisk list
DEVICE       TYPE           DISK    GROUP   STATUS
c0t06d0      auto:cdsdisk   -       -       online
c0t16d0      auto:cdsdisk   -       -       online
...
c2t64d0      auto:cdsdisk   -       -       online
c2t65d0      auto:cdsdisk   -       -       online
c2t66d0      auto:cdsdisk   -       -       online udid_mismatch
c2t67d0      auto:cdsdisk   -       -       online udid_mismatch
c2t68d0      auto:cdsdisk   -       -       online udid_mismatch
```

Writing a new UDID to a disk

You can use the following command to update the unique disk identifier (UDID) for one or more disks:

```
# vxdisk [-f] [-g diskgroup] updateudid disk ...
```

This command uses the current value of the UDID that is stored in the Device Discovery Layer (DDL) database to correct the value in the private region. The -f option must be specified if VxVM has not set the udid_mismatch flag for a disk.

For example, the following command updates the UDIDs for the disks c2t66d0 and c2t67d0:

```
# vxdisk updateudid c2t66d0 c2t67d0
```

Importing a disk group containing cloned disks

By default, disks on which the udid_mismatch flag or the clone_disk flag has been set are not imported by the vxdg import command unless all disks in the disk group have at least one of these flags set, and no two of the disks have the same UDID. You can then import the cloned disks by specifying the -o useclonedev=on option to the vxdg import command, as shown in this example:

```
# vxdg -o useclonedev=on [-o updateid] import mydg
```

Note: This form of the command allows only cloned disks to be imported. All non-cloned disks remain unimported.

If the `clone_disk` flag is set on a disk, this indicates the disk was previously imported into a disk group with the `udid_mismatch` flag set.

The `-o updateid` option can be specified to write new identification attributes to the disks, and to set the `clone_disk` flag on the disks. (The `vxdisk set clone=on` command can also be used to set the flag.) However, the import fails if multiple copies of one or more cloned disks exist. In this case, you can either update the UDIDs of the cloned disks as described in "Writing a new UDID to a disk" on page 176, or you can use the following command to tag all the disks in the disk group that are to be imported:

```
# vxdisk [-g diskgroup] settag tagname disk ...
```

where *tagname* is a string of up to 128 characters, not including spaces or tabs. For example, the following command sets the tag, `my_tagged_disks`, on several disks that are to be imported together:

```
# vxdisk settag my_tagged_disks c2t66d0 c2t67d0
```

To check which disks are tagged, use the `vxdisk listtag` command:

```
# vxdisk listtag
DANAME       DMNAME       NAME                VALUE
c0t06d0      mydg01       -                   -
c0t16d0      mydg02       -                   -
...
c2t64d0      mydg05       my_tagged_disks     -
c2t65d0      mydg06       my_tagged_disks     -
c2t66d0      mydg07       my_tagged_disks     -
c2t67d0      mydg08       my_tagged_disks     -
c2t68d0      mydg09       -                   -
```

The configuration database in a VM disk's private region contains persistent configuration data (or metadata) about the objects in a disk group. This database is consulted by VxVM when the disk group is imported. At least one of the cloned disks that are being imported must contain a copy of the current configuration database in its private region.

You can use the following command to ensure that a copy of the metadata is placed on a disk, regardless of the placement policy for the disk group:

```
# vxdisk [-g diskgroup] set disk keepmeta=always
```

Alternatively, use the following command to place a copy of the configuration database and kernel log on all disks in a disk group that share a given tag:

```
# vxdg [-g diskgroup] set tagmeta=on tag=tagname nconfig=all \
  nlog=all
```

To check which disks in a disk group contain copies of this configuration information, use the `vxdg listmeta` command:

```
# vxdg [-q] listmeta diskgroup
```

The `-q` option can be specified to suppress detailed configuration information from being displayed.

The tagged disks in the disk group may be imported by specifying the tag to the `vxdg import` command in addition to the `-o useclonedev=on` option:

```
# vxdg -o useclonedev=on -o tag=my_tagged_disks import mydg
```

If you have already imported the non-cloned disks in a disk group, you can use the `-n` and `-t` option to specify a temporary name for the disk group containing the cloned disks:

```
# vxdg -t -n clonedg -o useclonedev=on -o tag=my_tagged_disks \
  import mydg
```

See "Renaming a disk group" on page 183 for more information.

To remove a tag from a disk, use the `vxdisk rmtag` command, as shown in the following example:

```
# vxdisk rmtag tag=my_tagged_disks c2t67d0
```

For further information about the use of the `vxdisk` and `vxdg` commands to tag disks, and handle duplicate UDIDs, see the `vxdisk`(1M) and `vxdg`(1M) manual pages.

Sample cases of operations on cloned disks

The following sections contain examples of operations that can be used with cloned disks:

- Enabling configuration database copies on tagged disks
- Importing cloned disks without tags
- Importing cloned disks with tags

Enabling configuration database copies on tagged disks

In this example, the following commands have been used to tag some of the disks in an Hitachi TagmaStore array:

```
# vxdisk settag TagmaStore-USP0_28 t1=v1
# vxdisk settag TagmaStore-USP0_28 t2=v2
# vxdisk settag TagmaStore-USP0_24 t2=v2
# vxdisk settag TagmaStore-USP0_25 t1=v1
```

These tags can be viewed by using the `vxdisk listtag` command:

```
# vxdisk listtag
DEVICE              NAME    VALUE
TagmaStore-USP0_24  t2      v2
TagmaStore-USP0_25  t1      v1
TagmaStore-USP0_28  t1      v1
TagmaStore-USP0_28  t2      v2
```

The following command ensures that configuration database copies and kernel log copies are maintained for all disks in the disk group `mydg` that are tagged as `t1`:

```
# vxdg -g mydg set tagmeta=on tag=t1 nconfig=all nlog=all
```

The disks for which such metadata is maintained can be seen by using this command:

```
# vxdisk -o alldgs list
DEVICE             TYPE          DISK    GROUP  STATUS
TagmaStore-USP0_10 auto:cdsdisk -       -      online
TagmaStore-USP0_24 auto:cdsdisk mydg02  mydg   online
TagmaStore-USP0_25 auto:cdsdisk mydg03  mydg   online tagmeta
TagmaStore-USP0_26 auto:cdsdisk -       -      online
TagmaStore-USP0_27 auto:cdsdisk -       -      online
TagmaStore-USP0_28 auto:cdsdisk mydg01  mydg   online tagmeta
```

Alternatively, the following command can be used to ensure that a copy of the metadata is kept with a disk:

```
# vxdisk set TagmaStore-USP0_25 keepmeta=always
# vxdisk -o alldgs list
DEVICE             TYPE          DISK    GROUP  STATUS
TagmaStore-USP0_10 auto:cdsdisk -       -      online
TagmaStore-USP0_22 auto:cdsdisk -       -      online
TagmaStore-USP0_23 auto:cdsdisk -       -      online
TagmaStore-USP0_24 auto:cdsdisk mydg02  mydg   online
TagmaStore-USP0_25 auto:cdsdisk mydg03  mydg   online keepmeta
TagmaStore-USP0_28 auto:cdsdisk mydg01  mydg   online
```

Importing cloned disks without tags

In the first example, cloned disks (shadow image devices) from an Hitachi TagmaStore array are to be imported. The primary (non-cloned) disks, `mydg01`, `mydg02` and `mydg03`, are already imported, and the cloned disks are not tagged.

```
# vxdisk -o alldgs list
DEVICE             TYPE          DISK    GROUP  STATUS
TagmaStore-USP0_3  auto:cdsdisk -       (mydg) online udid_mismatch
TagmaStore-USP0_23 auto:cdsdisk mydg02   mydg  online
TagmaStore-USP0_25 auto:cdsdisk mydg03   mydg  online
TagmaStore-USP0_30 auto:cdsdisk -       (mydg) online udid_mismatch
TagmaStore-USP0_31 auto:cdsdisk -       (mydg) online udid_mismatch
TagmaStore-USP0_32 auto:cdsdisk mydg01   mydg  online
```

To import the cloned disks, they must be assigned a new disk group name, and their UDIDs must be updated:

```
# vxdg -n newdg -o useclonedev=on -o updateid import mydg
# vxdisk -o alldgs list
DEVICE             TYPE          DISK    GROUP  STATUS
TagmaStore-USP0_3  auto:cdsdisk mydg03  newdg  online clone_disk
TagmaStore-USP0_23 auto:cdsdisk mydg02  mydg   online
TagmaStore-USP0_25 auto:cdsdisk mydg03  mydg   online
TagmaStore-USP0_30 auto:cdsdisk mydg02  newdg  online clone_disk
TagmaStore-USP0_31 auto:cdsdisk mydg01  newdg  online clone_disk
TagmaStore-USP0_32 auto:cdsdisk mydg01  mydg   online
```

Note that the state of the imported cloned disks has changed from `online udid_mismatch` to `online clone_disk`.

In the next example, none of the disks (neither cloned nor non-cloned) have been imported:

```
# vxdisk -o alldgs list
DEVICE             TYPE          DISK    GROUP  STATUS
TagmaStore-USP0_3  auto:cdsdisk -       (mydg) online udid_mismatch
TagmaStore-USP0_23 auto:cdsdisk -       (mydg) online
TagmaStore-USP0_25 auto:cdsdisk -       (mydg) online
TagmaStore-USP0_30 auto:cdsdisk -       (mydg) online udid_mismatch
TagmaStore-USP0_31 auto:cdsdisk -       (mydg) online udid_mismatch
TagmaStore-USP0_32 auto:cdsdisk -       (mydg) online
```

To import only the cloned disks into the `mydg` disk group:

```
# vxdg -o useclonedev=on -o updateid import mydg
# vxdisk -o alldgs list
DEVICE             TYPE          DISK    GROUP  STATUS
TagmaStore-USP0_3  auto:cdsdisk mydg03  mydg   online clone_disk
TagmaStore-USP0_23 auto:cdsdisk -       (mydg) online
TagmaStore-USP0_25 auto:cdsdisk -       (mydg) online
TagmaStore-USP0_30 auto:cdsdisk mydg02  mydg   online clone_disk
TagmaStore-USP0_31 auto:cdsdisk mydg01  mydg   online clone_disk
TagmaStore-USP0_32 auto:cdsdisk -       (mydg) online
```

In the next example, a cloned disk (BCV device) from an EMC Symmetrix DMX array is to be imported. Before the cloned disk, `EMC0_27`, has been split off from the disk group, the `vxdisk list` command shows that it is in the `error udid_mismatch` state:

```
# vxdisk -o alldgs list
DEVICE             TYPE          DISK    GROUP  STATUS
EMC0_1             auto:cdsdisk EMC0_1  mydg   online
EMC0_27            auto          -       -      error udid_mismatch
```

The `symmir` command is used to split off the BCV device:

```
# /usr/symcli/bin/symmir -g mydg split DEV001
```

After updating VxVM's information about the disk by running the `vxdisk scandisks` command, the cloned disk is in the `online udid_mismatch` state:

```
# vxdisk -o alldgs list
```

```
DEVICE             TYPE         DISK    GROUP  STATUS
EMC0_1             auto:cdsdisk EMC0_1  mydg   online
EMC0_27            auto:cdsdisk -       -      online udid_mismatch
```

The following command imports the cloned disk into the new disk group `newdg`, and updates the disk's UDID:

```
# vxdg -n newdg -o useclonedev=on -o updateid import mydg
```

The state of the cloned disk is now shown as `online clone_disk`:

```
# vxdisk -o alldgs list
DEVICE             TYPE         DISK    GROUP  STATUS
EMC0_1             auto:cdsdisk EMC0_1  mydg   online
EMC0_27            auto:cdsdisk EMC0_1  newdg  online clone_disk
```

Importing cloned disks with tags

In this example, cloned disks (BCV devices) from an EMC Symmetrix DMX array are to be imported. The primary (non-cloned) disks, `mydg01`, `mydg02` and `mydg03`, are already imported, and the cloned disks with the tag `t1` are to be imported.

```
# vxdisk -o alldgs list
DEVICE             TYPE         DISK    GROUP  STATUS
EMC0_4             auto:cdsdisk mydg01  mydg   online
EMC0_6             auto:cdsdisk mydg02  mydg   online
EMC0_8             auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_15            auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_18            auto:cdsdisk mydg03  mydg   online
EMC0_24            auto:cdsdisk -       (mydg) online udid_mismatch
```

The disks are tagged as follows:

```
# vxdisk listtag
DEVICE             NAME    VALUE
EMC0_4             t2      v2
EMC0_4             t1      v1
EMC0_6             t2      v2
EMC0_8             t1      v1
EMC0_15            t2      v2
EMC0_18            t1      v1
EMC0_24            t1      v1
EMC0_24            t2      v2
```

To import the cloned disks that are tagged as `t1`, they must be assigned a new disk group name, and their UDIDs must be updated:

```
# vxdg -n newdg -o useclonedev=on -o tag=t1 -o updateid import mydg
# vxdisk -o alldgs list
DEVICE             TYPE         DISK    GROUP  STATUS
EMC0_4             auto:cdsdisk mydg01  mydg   online
EMC0_6             auto:cdsdisk mydg02  mydg   online
EMC0_8             auto:cdsdisk mydg03  newdg  online clone_disk
EMC0_15            auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_18            auto:cdsdisk mydg03  mydg   online
EMC0_24            auto:cdsdisk mydg01  newdg  online clone_disk
```

As the cloned disk EMC0_15 is not tagged as t1, it is not imported. Note that the state of the imported cloned disks has changed from online udid_mismatch to online clone_disk.

By default, the state of imported cloned disks is shown as online clone_disk. This can be removed by using the vxdisk set command as shown here:

```
# vxdisk set EMC0_8 clone=off
# vxdisk -o alldgs list
DEVICE              TYPE         DISK    GROUP  STATUS
EMC0_4              auto:cdsdisk mydg01  mydg   online
EMC0_6              auto:cdsdisk mydg02  mydg   online
EMC0_8              auto:cdsdisk mydg03  newdg  online
EMC0_15             auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_18             auto:cdsdisk mydg03  mydg   online
EMC0_24             auto:cdsdisk mydg01  newdg  online clone_disk
```

In the next example, none of the disks (neither cloned nor non-cloned) have been imported:

```
# vxdisk -o alldgs list
DEVICE              TYPE         DISK    GROUP  STATUS
EMC0_4              auto:cdsdisk -       (mydg) online
EMC0_6              auto:cdsdisk -       (mydg) online
EMC0_8              auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_15             auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_18             auto:cdsdisk -       (mydg) online
EMC0_24             auto:cdsdisk -       (mydg) online udid_mismatch
```

To import only the cloned disks that have been tagged as t1 into the mydg disk group:

```
# vxdg -o useclonedev=on -o tag=t1 -o updateid import mydg
# vxdisk -o alldgs list
DEVICE              TYPE         DISK    GROUP  STATUS
EMC0_4              auto:cdsdisk -       (mydg) online
EMC0_6              auto:cdsdisk -       (mydg) online
EMC0_8              auto:cdsdisk mydg03  mydg   online clone_disk
EMC0_15             auto:cdsdisk -       (mydg) online udid_mismatch
EMC0_18             auto:cdsdisk -       (mydg) online
EMC0_24             auto:cdsdisk mydg01  mydg   online clone_disk
```

As in the previous example, the cloned disk EMC0_15 is not tagged as t1, and so it is not imported.

Renaming a disk group

Only one disk group of a given name can exist per system. It is not possible to import or deport a disk group when the target system already has a disk group of the same name. To avoid this problem, VxVM allows you to rename a disk group during import or deport.

To rename a disk group during import, use the following command:

```
# vxdg [-t] -n newdg import diskgroup
```

If the `-t` option is included, the import is temporary and does not persist across reboots. In this case, the stored name of the disk group remains unchanged on its original host, but the disk group is known by the name specified by *`newdg`* to the importing host. If the `-t` option is not used, the name change is permanent.

For example, this command temporarily renames the disk group, `mydg`, as `mytempdg` on import:

```
# vxdg -t -n mytempdg import mydg
```

To rename a disk group during deport, use the following command:

```
# vxdg [-h hostname] -n newdg deport diskgroup
```

When renaming on deport, you can specify the `-h` *hostname* option to assign a lock to an alternate host. This ensures that the disk group is automatically imported when the alternate host reboots.

For example, this command renames the disk group, `mydg`, as `myexdg`, and deports it to the host, `jingo`:

```
# vxdg -h jingo -n myexdg deport mydg
```

Note: You cannot use this method to rename the active boot disk group because it contains volumes that are in use by mounted file systems (such as `/`). To rename the boot disk group, boot the system from an LVM root disk instead of from the VxVM root disk. You can then use the above methods to rename the boot disk group. See the sections under "Rootability" on page 102 for more information.

To temporarily move the boot disk group, `bootdg`, from one host to another (for repair work on the root volume, for example) and then move it back

1 On the original host, identify the disk group ID of the `bootdg` disk group to be imported with the following command:

```
# vxdisk -g bootdg -s list
```

This command results in output such as the following:

```
dgname: rootdg
```

```
dgid:   774226267.1025.tweety
```

Note: In this example, the administrator has chosen to name the boot disk group as `rootdg`. The ID of this disk group is `774226267.1025.tweety`.

This procedure assumes that all the disks in the boot disk group are accessible by both hosts.

2 Shut down the original host.

3 On the importing host, import and rename the `rootdg` disk group with this command:

```
# vxdg -tC -n newdg import diskgroup
```

The `-t` option indicates a temporary import name, and the `-C` option clears import locks. The `-n` option specifies an alternate name for the `rootdg` being imported so that it does not conflict with the existing `rootdg`. *diskgroup* is the disk group ID of the disk group being imported (for example, `774226267.1025.tweety`).
If a reboot or crash occurs at this point, the temporarily imported disk group becomes unimported and requires a reimport.

4 After the necessary work has been done on the imported disk group, deport it back to its original host with this command:

```
# vxdg -h hostname deport diskgroup
```

Here *hostname* is the name of the system whose `rootdg` is being returned (the system name can be confirmed with the command `uname -n`).
This command removes the imported disk group from the importing host and returns locks to its original host. The original host can then automatically import its boot disk group at the next reboot.

Moving disks between disk groups

To move a disk between disk groups, remove the disk from one disk group and add it to the other. For example, to move the physical disk `c0t3d0` (attached with the disk name `salesdg04`) from disk group `salesdg` and add it to disk group `mktdg`, use the following commands:

```
# vxdg -g salesdg rmdisk salesdg04
# vxdg -g mktdg adddisk mktdg02=c0t3d0
```

Caution: This procedure does not save the configurations nor data on the disks.

You can also move a disk by using the `vxdiskadm` command. Select item `3 (Remove a disk)` from the main menu, and then select item `1 (Add or initialize a disk)`.

See "Moving objects between disk groups" on page 203 for an alternative and preferred method of moving disks between disk groups. This method preserves VxVM objects, such as volumes, that are configured on the disks.

Moving disk groups between systems

An important feature of disk groups is that they can be moved between systems. If all disks in a disk group are moved from one system to another, then the disk group can be used by the second system. You do not have to re-specify the configuration.

To move a disk group between systems

1. On the first system, stop all volumes in the disk group, then deport (disable local access to) the disk group with the following command:

   ```
   # vxdg deport diskgroup
   ```

2. Move all the disks to the second system and perform the steps necessary (system-dependent) for the second system and VxVM to recognize the new disks.

 This can require a reboot, in which case the `vxconfigd` daemon is restarted and recognizes the new disks. If you do not reboot, use the command `vxdctl enable` to restart the `vxconfigd` program so VxVM also recognizes the disks.

3. Import (enable local access to) the disk group on the second system with this command:

   ```
   # vxdg import diskgroup
   ```

 Caution: All disks in the disk group must be moved to the other system. If they are not moved, the import fails.

4. After the disk group is imported, start all volumes in the disk group with this command:

   ```
   # vxrecover -g diskgroup -sb
   ```

You can also move disks from a system that has crashed. In this case, you cannot deport the disk group from the first system. When a disk group is created or imported on a system, that system writes a lock on all disks in the disk group.

Caution: The purpose of the lock is to ensure that *dual-ported disks* (disks that can be accessed simultaneously by two systems) are not used by both systems at the same time. If two systems try to access the same disks at the same time, this must be managed using software such as the clustering functionality of VxVM. Otherwise, configuration information stored on the disk may be corrupted, and the data on the disk may become unusable.

Handling errors when importing disks

When you move disks from a system that has crashed or that failed to detect the group before the disk was moved, the locks stored on the disks remain and must be cleared. The system returns the following error message:

```
VxVM vxdg ERROR V-5-1-587 disk group groupname: import failed:
Disk is in use by another host
```

The next message indicates that the disk group does not contains any valid disks (not that it does not contains any disks):

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
No valid disk found containing disk group
```

The disks may be considered invalid due to a mismatch between the host ID in their configuration copies and that stored in the `/etc/vx/volboot` file.

To clear locks on a specific set of devices, use the following command:

```
# vxdisk clearimport devicename ...
```

To clear the locks during import, use the following command:

```
# vxdg -C import diskgroup
```

Caution: Be careful when using the `vxdisk clearimport` or `vxdg -C import` command on systems that have dual-ported disks. Clearing the locks allows those disks to be accessed at the same time from multiple hosts and can result in corrupted data.

A disk group can be imported successfully if all the disks are accessible that were visible when the disk group was last imported successfully. However, sometimes you may need to specify the `-f` option to forcibly import a disk group if some disks are not available. If the `import` operation fails, an error message is displayed.

The following error message indicates a fatal error that requires hardware repair or the creation of a new disk group, and recovery of the disk group configuration and data:

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
Disk group has no valid configuration copies
```

The following error message indicates a recoverable error.

```
VxVM vxdg ERROR V-5-1-587 Disk group groupname: import failed:
Disk for disk group not found
```

If some of the disks in the disk group have failed, you can force the disk group to be imported by specifying the `-f` option to the `vxdg import` command:

```
# vxdg -f import diskgroup
```

Caution: Be careful when using the `-f` option. It can cause the same disk group to be imported twice from different sets of disks. This can cause the disk group configuration to become inconsistent.
See "Handling conflicting configuration copies" on page 190.

As using the `-f` option to force the import of an incomplete disk group counts as a successful import, an incomplete disk group may be imported subsequently without this option being specified. This may not be what you expect.

These operations can also be performed using the `vxdiskadm` utility. To deport a disk group using `vxdiskadm`, select menu item `8 (Remove access to (deport) a disk group)`. To import a disk group, select item `7 (Enable access to (import) a disk group)`. The `vxdiskadm` import operation checks for host import locks and prompts to see if you want to clear any that are found. It also starts volumes in the disk group.

Reserving minor numbers for disk groups

A *device minor number* uniquely identifies some characteristic of a device to the device driver that controls that device. It is often used to identify some characteristic mode of an individual device, or to identify separate devices that are all under the control of a single controller. VxVM assigns unique device minor numbers to each object (volume, plex, subdisk, disk, or disk group) that it controls.

When you move a disk group between systems, it is possible for the minor numbers that it used on its previous system to coincide (or *collide*) with those of objects known to VxVM on the new system. To get around this potential problem, you can allocate separate ranges of minor numbers for each disk group. VxVM uses the specified range of minor numbers when it creates volume objects from the disks in the disk group. This guarantees that each volume has the same minor number across reboots or reconfigurations. Disk groups may then be moved between machines without causing device number collisions.

VxVM chooses minor device numbers for objects created from this disk group starting at the base minor number *base_minor*. Minor numbers can range from this value up to 16,777,215. Try to leave a reasonable number of unallocated

minor numbers near the top of this range to allow for temporary device number remapping in the event that a device minor number collision may still occur.

VxVM reserves the range of minor numbers from 0 to 999 for use with volumes in the boot disk group. For example, the `rootvol` volume is always assigned minor number 0.

If you do not specify the base of the minor number range for a disk group, VxVM chooses one at random. The number chosen is at least 1000, is a multiple of 1000, and yields a usable range of 1000 device numbers. The chosen number also does not overlap within a range of 1000 of any currently imported disk groups, and it does not overlap any currently allocated volume device numbers.

Note: The default policy ensures that a small number of disk groups can be merged successfully between a set of machines. However, where disk groups are merged automatically using failover mechanisms, select ranges that avoid overlap.

To view the base minor number for an existing disk group, use the `vxprint` command as shown in the following examples for the disk group, `mydg`:

```
# vxprint -l mydg | egrep minors
minors: >=45000
# vxprint -g mydg -m | egrep base_minor
base_minor=45000
```

To set a base volume device minor number for a disk group that is being created, use the following command:

```
# vxdg init diskgroup minor=base_minor disk_access_name ...
```

For example, the following command creates the disk group, `newdg`, that includes the specified disks, and has a base minor number of 30000:

```
# xvdg init newdg minor=30000 c1d0t0 c1t1d0
```

If a disk group already exists, you can use the `vxdg reminor` command to change its base minor number:

```
# vxdg -g diskgroup reminor new_base_minor
```

For example, the following command changes the base minor number to 30000 for the disk group, `mydg`:

```
# vxprint -g mydg reminor 30000
```

If a volume is open, its old device number remains in effect until the system is rebooted or until the disk group is deported and re-imported. If you close the open volume, you can run `vxdg reminor` again to allow the renumbering to take effect without rebooting or re-importing.

An example of where it is necessary to change the base minor number is for a cluster-shareable disk group. The volumes in a shared disk group must have the same minor number on all the nodes. If there is a conflict between the minor numbers when a node attempts to join the cluster, the join fails. You can use the

reminor operation on the nodes that are in the cluster to resolve the conflict. In a cluster where more than one node is joined, use a base minor number which does not conflict on any node.

For further information on minor number reservation, see the vxdg(1M) manual page.

Compatibility of disk groups between platforms

For disk groups that support the Cross-platform Data Sharing (CDS) feature, the upper limit on the minor number range is restricted on AIX, HP-UX, Linux (with a 2.6 or later kernel) and Solaris to 65,535 to ensure portability between these operating systems.

On a Linux platform with a pre-2.6 kernel, the number of minor numbers per major number is limited to 256 with a base of 0. This has the effect of limiting the number of volumes and disks that can be supported system-wide to a smaller value than is allowed on other operating system platforms. The number of disks that are supported by a pre-2.6 Linux kernel is typically limited to a few hundred. With the extended major numbering scheme that was implemented in VxVM 4.0 on Linux, a maximum of 4079 volumes could be configured, provided that a contiguous block of 15 extended major numbers was available.

VxVM 4.1 runs on a 2.6 version Linux kernel, which increases the number of minor devices that are configurable from 256 to 65,536 per major device number. This allows a large number of volumes and disk devices to be configured on a system. The theoretical limit on the number of DMP and volume devices that can be configured on such a system are 65,536 and 1,048,576 respectively. However, in practice, the number of VxVM devices that can be configured in a single disk group is limited by the size of the private region.

When a CDS-compatible disk group is imported on a Linux system with a pre-2.6 kernel, VxVM attempts to reassign the minor numbers of the volumes, and fails if this is not possible.

To help ensure that a CDS-compatible disk group is portable between operating systems, including Linux with a pre-2.6 kernel, use the following command to set the maxdev attribute on the disk group:

```
# vxdg -g diskgroup set maxdev=4079
```

Note: Such a disk group may still not be importable by VxVM 4.0 on Linux with a pre-2.6 kernel if it would increase the number of minor numbers on the system that are assigned to volumes to more than 4079, or if the number of available extended major numbers is smaller than 15.

You can use the following command to discover the maximum number of volumes that are supported by VxVM on a Linux host:

```
# cat /proc/sys/vxvm/vxio/vol_max_volumes
4079
```

See the vxdg(1M) manual page for more information.

Handling conflicting configuration copies

If an incomplete disk group is imported on several different systems, this can create inconsistencies in the disk group configuration copies that you may need to resolve manually. This section and following sections describe how such a condition can occur, and how to correct it. (When the condition occurs in a cluster that has been split, it is usually referred to as a *serial split brain* condition).

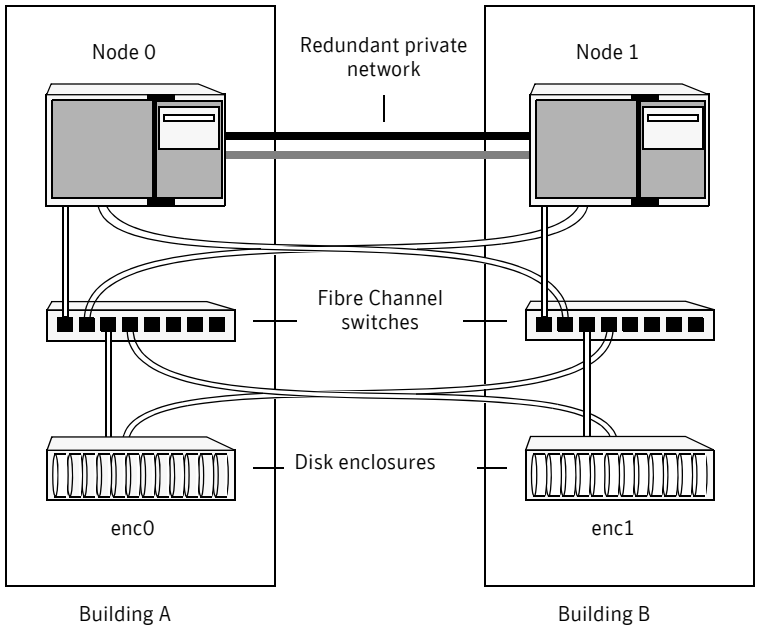
Note: The procedures given here require that the version number of the disk group is at least 110.

Example of a serial split brain condition in a cluster

Note: This section presents an example of how a serial split brain condition might occur for a shared disk group in a cluster. For more information about shared disk groups in clusters, see "Administering cluster functionality" on page 397. Conflicts between configuration copies can also occur for private disk groups in clustered and non-clustered configurations where the disk groups have been partially imported on different systems. The procedure in "Correcting conflicting configuration information" on page 194 describes how to correct such problems.

A campus cluster (also known as a *stretch cluster* or remote mirror configuration) typically consists of a 2-node cluster where each component (server, switch and storage) of the cluster exists in a separate building. Figure 4-1 illustrates a 2-node cluster with node 0, a fibre channel switch and disk enclosure enc0 in building A, and node 1, another switch and enclosure enc1 in building B. The fibre channel connectivity is multiply redundant to implement redundant-loop access between each node and each enclosure. As usual, the two nodes are also linked by a redundant private network.

Figure 4-1 Typical arrangement of a 2-node campus cluster



A serial split brain condition typically arises in a cluster when a private (non-shared) disk group is imported on Node 0 with Node 1 configured as the failover node.

If the network connections between the nodes are severed, both nodes think that the other node has died. (This is the usual cause of the split brain condition in clusters). If a disk group is spread across both enclosure `enc0` and `enc1`, each portion loses connectivity to the other portion of the disk group. Node 0 continues to update to the disks in the portion of the disk group that it can access. Node 1, operating as the failover node, imports the other portion of the disk group (with the `-f` option set), and starts updating the disks that it can see.

When the network links are restored, attempting to reattach the missing disks to the disk group on Node 0, or to re-import the entire disk group on either node, fails. This serial split brain condition arises because VxVM increments the serial ID in the disk media record of each imported disk in all the disk group configuration databases on those disks, and also in the private region of each imported disk. The value that is stored in the configuration database represents the serial ID that the disk group expects a disk to have. The serial ID that is stored in a disk's private region is considered to be its actual value.

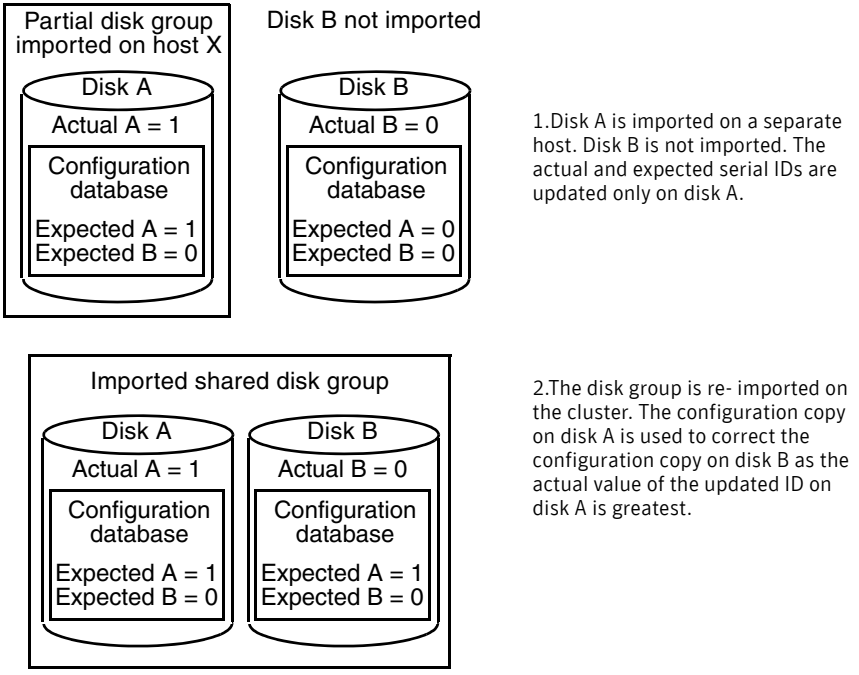
If some disks went missing from the disk group (due to physical disconnection or power failure) and those disks were imported by another host, the serial IDs

for the disks in their copies of the configuration database, and also in each disk's private region, are updated separately on that host. When the disks are subsequently re-imported into the original shared disk group, the actual serial IDs on the disks do not agree with the expected values from the configuration copies on other disks in the disk group.

Depending on what happened to the different portions of the split disk group, there are two possibilities for resolving inconsistencies between the configuration databases:

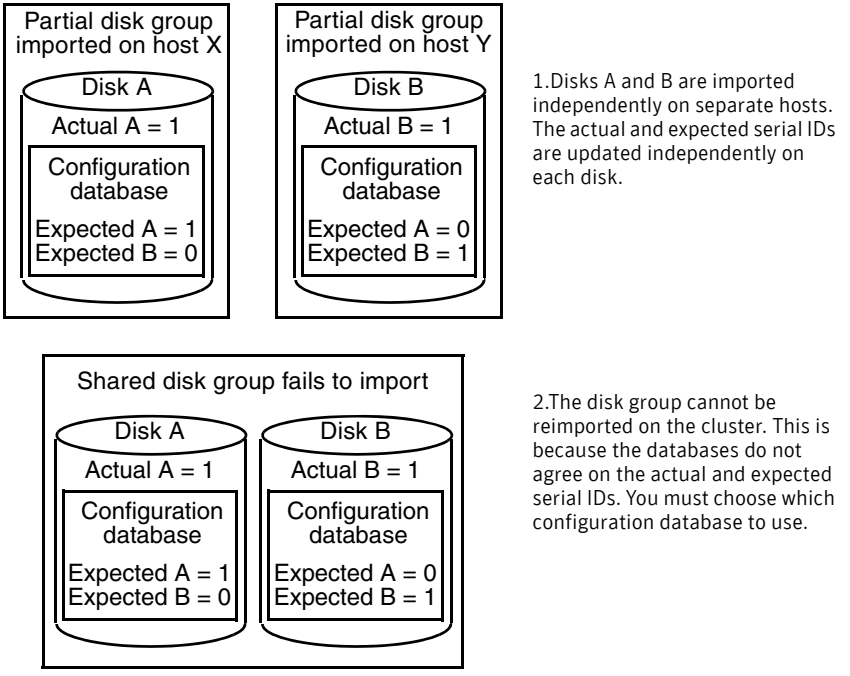
- If the other disks in the disk group were not imported on another host, VxVM resolves the conflicting values of the serial IDs by using the version of the configuration database from the disk with the greatest value for the updated ID (shown as the value of `update_tid` in the output from the `vxprint -m` *diskgroup* `| grep update_tid` command). This case is illustrated below.

Figure 4-2 Example of a serial split brain condition that can be resolved automatically



- If the other disks were also imported on another host, no disk can be considered to have a definitive copy of the configuration database. The figure below illustrates how this condition can arise for two disks.

Figure 4-3 Example of a true serial split brain condition that cannot be resolved automatically



This is a true serial split brain condition, which VxVM cannot correct automatically. In this case, the disk group import fails, and the `vxdg` utility outputs error messages similar to the following before exiting:

```
VxVM vxconfigd NOTICE V-5-0-33 Split Brain. da id is 0.1, while
dm id is 0.0 for DM mydg01
VxVM vxdg ERROR V-5-1-587 Disk group newdg: import failed:
Serial Split Brain detected. Run vxsplitlines
```

The import does not succeed even if you specify the `-f` flag to `vxdg`.

Although it is usually possible to resolve this conflict by choosing the version of the configuration database with the highest valued configuration ID (shown as the value of `seqno` in the output from the `vxdg list` *diskgroup* `| grep config` command), this may not be the correct thing to do in all circumstances.

The following section, "Correcting conflicting configuration information," describes how to fix this condition.

For more information on how to set up and maintain a remote mirror configuration, see "Administering sites and remote mirrors" on page 431.

Correcting conflicting configuration information

Note: This procedure requires that the disk group has a version number of at least 110. See "Upgrading a disk group" on page 208 for more information about disk group version numbers.

To resolve conflicting configuration information, you must decide which disk contains the correct version of the disk group configuration database. To assist you in doing this, you can run the `vxsplitlines` command to show the actual serial ID on each disk in the disk group and the serial ID that was expected from the configuration database. For each disk, the command also shows the `vxdg` command that you must run to select the configuration database copy on that disk as being the definitive copy to use for importing the disk group.

The following is sample output from running `vxsplitlines` on the disk group `newdg`:

```
# vxsplitlines -g newdg
The following splits were found in disk group newdg
They are listed in da(dm) name pairs.

Pool 0.
       c2t5d0 ( c2t5d0 ), c2t6d0 ( c2t6d0 ),
The configuration from any of the disks in this split should appear
to be be the same.
To see the configuration from any of the disks in this split, run:
    /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c2t5d0
To import the dg with the configuration from this split, run:
    /usr/sbin/vxdg -o selectcp=1045852127.32.olancha import newdg
To get more information about this particular configuration, run:
    /usr/sbin/vxsplitlines -g newdg -c c2t5d0

Split 1.
c2t7d0 ( c2t7d0 ), c2t8d0 ( c2t8d0 ),
The configuration from any of the disks in this split should appear
to be be the same.
To see the configuration from any of the disks in this split, run:
    /etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c2t7d0
To import the dg with the configuration from this split, run:
    /usr/sbin/vxdg -o selectcp=1045852127.33.olancha import newdg
To get more information about this particular configuration, run:
    /usr/sbin/vxsplitlines -g newdg -c c2t7d0
```

In this example, the disk group has four disks, and is split so that two disks appear to be on each side of the split.

You can specify the `-c` option to `vxsplitlines` to print detailed information about each of the disk IDs from the configuration copy on a disk specified by its disk access name:

```
# vxsplitlines  -g newdg -c c2t6d0
DANAME(DMNAME)      || Actual SSB     || Expected SSB
c2t5d0( c2t5d0 ) || 0.1               || 0.0 ssb ids don't match
c2t6d0( c2t6d0 ) || 0.1               || 0.1 ssb ids match
c2t7d0( c2t7d0 ) || 0.1               || 0.1 ssb ids match
c2t8d0( c2t8d0 ) || 0.1               || 0.0 ssb ids don't match

Please note that even though some disks ssb ids might match
that does not necessarily mean that those disks' config copies
have all the changes. From some other configuration copies,
those disks' ssb ids might not match.
To see the configuration from this disk, run
/etc/vx/diag.d/vxprivutil dumpconfig /dev/vx/dmp/c2t6d0
```

Based on your knowledge of how the serial split brain condition came about, you must choose one disk's configuration to be used to import the disk group. For example, the following command imports the disk group using the configuration copy that is on side 0 of the split:

```
# /usr/sbin/vxdg -o selectcp=1045852127.32.olancha import newdg
```

When you have selected a preferred configuration copy, and the disk group has been imported, VxVM resets the serial IDs to 0 for the imported disks. The actual and expected serial IDs for any disks in the disk group that are not imported at this time remain unaltered.

Reorganizing the contents of disk groups

Note: You need a Veritas FlashSnap™ license to use this feature.

There are several circumstances under which you might want to reorganize the contents of your existing disk groups:

- To group volumes or disks differently as the needs of your organization change. For example, you might want to split disk groups to match the boundaries of separate departments, or to join disk groups when departments are merged.
- To reduce the size of a disk group's configuration database in the event that its private region is nearly full. This is a much simpler solution than the alternative of trying to grow the private region.

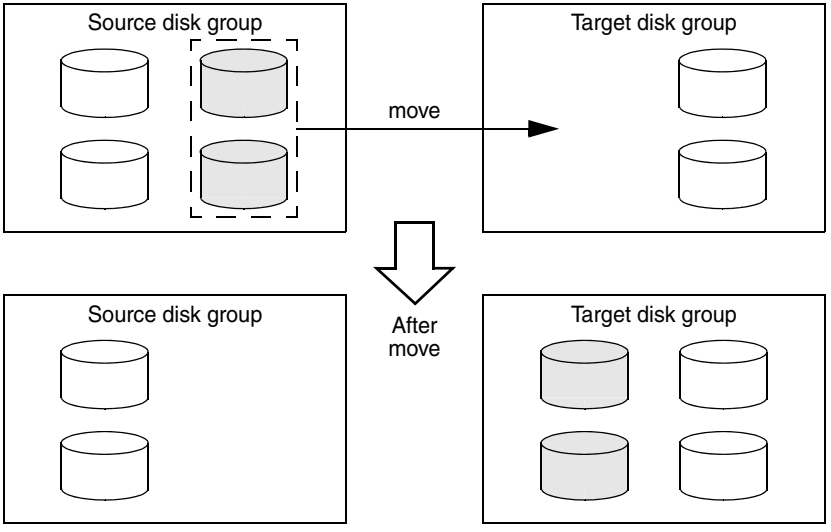
- To perform online maintenance and upgrading of fault-tolerant systems that can be split into separate hosts for this purpose, and then rejoined.
- To isolate volumes or disks from a disk group, and process them independently on the same host or on a different host. This allows you to implement off-host processing solutions for the purposes of backup or decision support. This is discussed further in "Configuring off-host processing" on page 369.

You can use either the Veritas Enterprise Administrator (VEA) or the `vxdg` command to reorganize your disk groups. For more information about using the graphical user interface, see the *Veritas Enterprise Administrator User's Guide* and VEA online help. This section describes how to use the `vxdg` command.

The `vxdg` command provides the following operations for reorganizing disk groups:

- `move`—moves a self-contained set of VxVM objects between imported disk groups. This operation fails if it would remove all the disks from the source disk group. Volume states are preserved across the move. The `move` operation is illustrated in Figure 4-4.

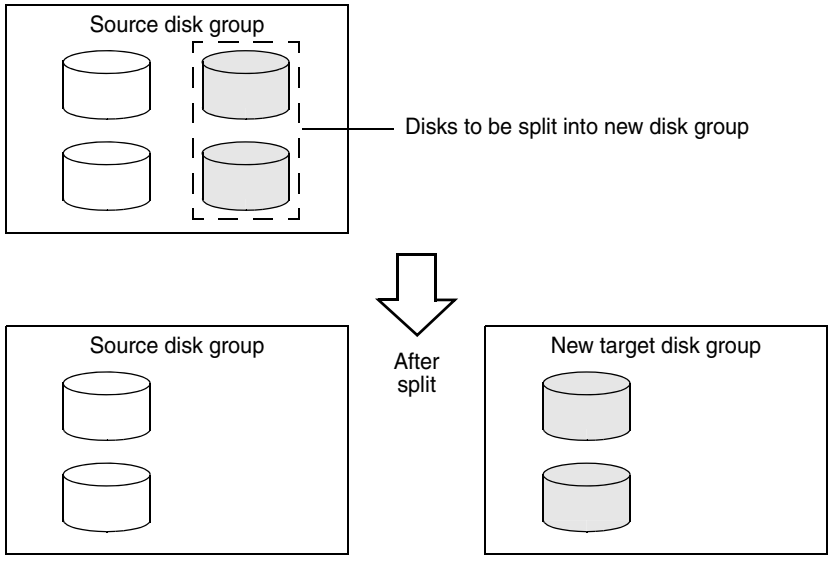
Figure 4-4 Disk group move operation



- `split`—removes a self-contained set of VxVM objects from an imported disk group, and moves them to a newly created target disk group. This operation fails if it would remove all the disks from the source disk group, or if an

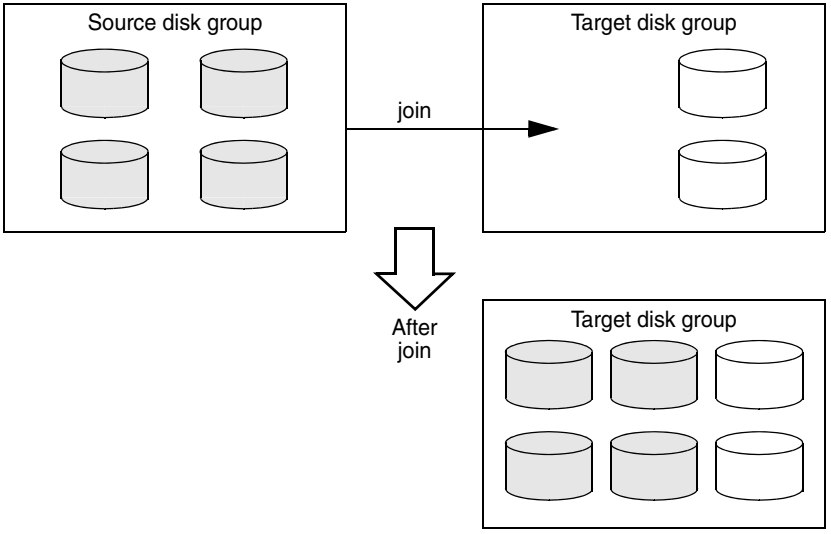
imported disk group exists with the same name as the target disk group. An existing deported disk group is destroyed if it has the same name as the target disk group (as is the case for the `vxdg init` command). The `split` operation is illustrated in Figure 4-5.

Figure 4-5 Disk group split operation



- `join`—removes all VxVM objects from an imported disk group and moves them to an imported target disk group. The source disk group is removed when the join is complete. The `join` operation is illustrated in Figure 4-6.

Figure 4-6 Disk group join operation



These operations are performed on VxVM objects such as disks or top-level volumes, and include all component objects such as sub-volumes, plexes and subdisks. The objects to be moved must be *self-contained*, meaning that the disks that are moved must not contain any other objects that are not intended for the move.

If you specify one or more disks to be moved, all VxVM objects on the disks are moved. You can use the `-o expand` option to ensure that `vxdg` moves all disks on which the specified objects are configured. Care should be taken when doing this as the result may not always be what you expect. You can use the `listmove` operation with `vxdg` to help you establish what is the self-contained set of objects that corresponds to a specified set of objects.

Caution: Before moving volumes between disk groups, stop all applications that are accessing the volumes, and unmount all file systems that are configured on these volumes.

If the system crashes or a hardware subsystem fails, VxVM attempts to complete or reverse an incomplete disk group reconfiguration when the system is restarted or the hardware subsystem is repaired, depending on how far the reconfiguration had progressed. If one of the disk groups is no longer available because it has been imported by another host or because it no longer exists, you

must recover the disk group manually as described in the section “Recovery from Incomplete Disk Group Moves” in the chapter “Recovery from Hardware Failure” of the *Veritas Volume Manager Troubleshooting Guide*.

Limitations of disk group split and join

The disk group split and join feature has the following limitations:

- Disk groups involved in a move, split or join must be version 90 or greater. See “Upgrading a disk group” on page 208 for more information on disk group versions.
- The reconfiguration must involve an integral number of physical disks.
- Objects to be moved must not contain open volumes.
- Disks cannot be moved between CDS and non-CDS compatible disk groups.
- Moved volumes are initially disabled following a disk group move, split or join. Use the `vxrecover -m` and `vxvol startall` commands to recover and restart the volumes.
- Data change objects (DCOs) and snap objects that have been dissociated by Persistent FastResync cannot be moved between disk groups.
- Veritas Volume Replicator (VVR) objects cannot be moved between disk groups.
- For a disk group move to succeed, the source disk group must contain at least one disk that can store copies of the configuration database after the move.
- For a disk group split to succeed, both the source and target disk groups must contain at least one disk that can store copies of the configuration database after the split.
- For a disk group move or join to succeed, the configuration database in the target disk group must be able to accommodate information about all the objects in the enlarged disk group.
- Splitting or moving a volume into a different disk group changes the volume’s record ID.
- The operation can only be performed on the master node of a cluster if either the source disk group or the target disk group is shared.
- In a cluster environment, disk groups involved in a move or join must both be private or must both be shared.
- When used with objects that have been created using the Veritas Intelligent Storage Provisioning (ISP) feature, only complete storage pools may be split or moved from a disk group. Individual objects such as application volumes

within storage pools may not be split or moved. See the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for a description of ISP and storage pools.

- If a cache object or volume set that is to be split or moved uses ISP volumes, the storage pool that contains these volumes must also be specified.

The following sections describe how to use the `vxdg` command to reorganize disk groups. For more information about the `vxdg` command, see the `vxdg`(1M) manual page.

Listing objects potentially affected by a move

To display the VxVM objects that would be moved for a specified list of objects, use the following command:

```
# vxdg [-o expand] listmove sourcedg targetdg object ...
```

The following example lists the objects that would be affected by moving volume `vol1` from disk group `mydg` to `newdg`:

```
# vxdg listmove mydg newdg vol1
mydg01 c0t1d0 mydg05 c1t96d0 vol1 vol1-01 vol1-02 mydg01-01
mydg05-01
```

However, the following command produces an error because only a part of the volume `vol1` is configured on the disk `mydg01`:

```
# vxdg listmove mydg newdg mydg01
VxVM vxdg ERROR V-5-2-4597 vxdg listmove mydg newdg failed
VxVM vxdg ERROR V-5-2-3091 mydg05 : Disk not moving, but
subdisks on it are
```

Specifying the `-o expand` option, as shown below, ensures that the list of objects to be moved includes the other disks (in this case, `mydg05`) that are configured in `vol1`:

```
# vxdg -o expand listmove mydg newdg mydg01
mydg01 c0t1d0 mydg05 c1t96d0 vol1 vol1-01 vol1-02 mydg01-01
mydg05-01
```

Moving DCO volumes between disk groups

When you move the parent volume (such as a snapshot volume) to a different disk group, its DCO volume must accompany it. If you use the `vxassist addlog`, `vxmake` or `vxdco` commands to set up a DCO for a volume, you must ensure that the disks that contain the plexes of the DCO volume accompany their parent volume during the move. You can use the `vxprint` command on a volume to examine the configuration of its associated DCO volume.

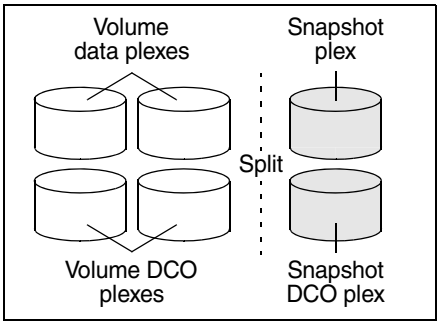
If you use the `vxassist` command or the Veritas Enterprise Administrator (VEA) to create both a volume and its DCO, or the `vxsnap prepare` command to add a DCO to a volume, the DCO plexes are automatically placed on different disks from the data plexes of the parent volume. In previous releases, version 0 DCO

plexes were placed on the same disks as the data plexes for convenience when performing disk group split and move operations. As version 20 DCOs support dirty region logging (DRL) in addition to Persistent FastResync, it is preferable for the DCO plexes to be separated from the data plexes. This improves the performance of I/O from/to the volume, and provides resilience for the DRL logs.

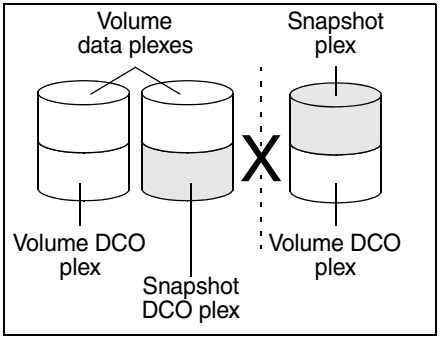
Figure 4-7 illustrates some instances in which it is not be possible to split a disk group because of the location of the DCO plexes on the disks of the disk group. For more information about relocating DCO plexes, see "Specifying storage for version 0 DCO plexes" on page 357 and "Specifying storage for version 20 DCO plexes" on page 276.

For more information about the layout of DCO volumes and their use with volume snapshots, see and "FastResync" on page 66. For more information about the administration of volume snapshots, see "Volume snapshots" on page 63 and "Administering volume snapshots" on page 303.

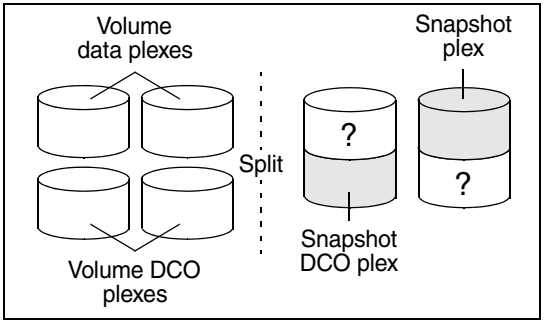
Figure 4-7 Examples of disk groups that can and cannot be split



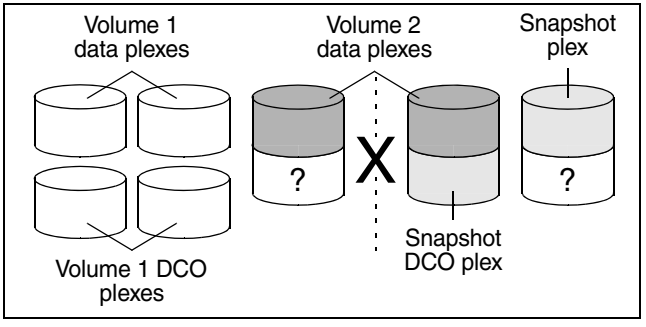
The disk group can be split as the DCO plexes are on dedicated disks, and can therefore accompany the disks that contain the volume data.



The disk group cannot be split as the DCO plexes cannot accompany their volumes. One solution is to relocate the DCO plexes. In this example, use an additional disk in the disk group as an intermediary to swap the misplaced DCO plexes. Alternatively, to improve DRL performance and resilience, allocate the DCO plexes to dedicated disks.



The disk group can be split as the DCO plexes can accompany their volumes. However, you may not wish the data in the portions of the disks marked "?" to be moved as well.



The disk group cannot be split as this would separate the disks containing Volume 2's data plexes. Possible solutions are to relocate the snapshot DCO plex to the snapshot plex disk, or to another suitable disk that can be moved.

Moving objects between disk groups

To move a self-contained set of VxVM objects from an imported source disk group to an imported target disk group, use the following command:

```
# vxdg [-o expand] [-o override|verify] move sourcedg targetdg \
    object ...
```

The `-o expand` option ensures that the objects that are actually moved include all other disks containing subdisks that are associated with the specified objects or with objects that they contain.

The default behavior of `vxdg` when moving licensed disks in an EMC array is to perform an EMC disk compatibility check for each disk involved in the move. If the compatibility checks succeed, the move takes place. `vxdg` then checks again to ensure that the configuration has not changed since it performed the compatibility check. If the configuration has changed, `vxdg` attempts to perform the entire move again.

The `-o override` option enables the move to take place without any EMC checking.

The `-o verify` option returns the access names of the disks that would be moved but does not perform the move.

Note: The `-o override` and `-o verify options` require a valid EMC license.

See "Moving objects between disk groups" on page 424 for information on how to move objects between disk groups in a cluster.

For example, the following output from `vxprint` shows the contents of disk groups `rootdg` and `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC        KSTATE    LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg rootdg       rootdg       -         -          -         -       -         -
dm rootdg02     c1t97d0      -         17678493   -         -       -         -
dm rootdg03     c1t112d0     -         1767849 3  -         -       -         -
dm rootdg04     c1t114d0     -         17678493   -         -       -         -
dm rootdg06     c1t98d0      -         17678493   -         -       -         -

Disk group: mydg
TY NAME         ASSOC        KSTATE    LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg mydg         mydg         -         -          -         -       -         -
dm mydg01       c0t1d0       -         17678493   -         -       -         -
dm mydg05       c1t96d0      -         17678493   -         -       -         -
dm mydg07       c1t99d0      -         17678493   -         -       -         -
dm mydg08       c1t100d0     -         17678493   -         -       -         -
v  vol1         fsgen        ENABLED   2048       -         ACTIVE  -         -
pl vol1-01      vol1         ENABLED   3591       -         ACTIVE  -         -
sd mydg01-01    vol1-01      ENABLED   3591       0         -       -         -
pl vol1-02      vol1         ENABLED   3591       -         ACTIVE  -         -
sd mydg05-01    vol1-02      ENABLED   3591       0         -       -         -
```

The following command moves the self-contained set of objects implied by specifying disk `mydg01` from disk group `mydg` to `rootdg`:

```
# vxdg -o expand move mydg rootdg mydg01
```

The moved volumes are initially disabled following the move. Use the following commands to recover and restart the volumes in the target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the move shows that not only `mydg01` but also volume `vol1` and `mydg05` have moved to `rootdg`, leaving only `mydg07` and `mydg08` in disk group `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC        KSTATE    LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg rootdg       rootdg       -         -          -         -       -         -
dm mydg01       c0t1d0       -         17678493   -         -       -         -
dm rootdg02     c1t97d0      -         17678493   -         -       -         -
dm rootdg03     c1t112d0     -         1767849 3  -         -       -         -
dm rootdg04     c1t114d0     -         17678493   -         -       -         -
dm mydg05       c1t96d0      -         17678493   -         -       -         -
dm rootdg06     c1t98d0      -         17678493   -         -       -         -
v  vol1         fsgen        ENABLED   2048       -         ACTIVE  -         -
pl vol1-01      vol1         ENABLED   3591       -         ACTIVE  -         -
sd mydg01-01    vol1-01      ENABLED   3591       0         -       -         -
pl vol1-02      vol1         ENABLED   3591       -         ACTIVE  -         -
sd mydg05-01    vol1-02      ENABLED   3591       0         -       -         -
```

```
Disk group: mydg
TY NAME         ASSOC        KSTATE    LENGTH      PLOFFS    STATE   TUTIL0    PUTIL0
dg mydg         mydg         -         -           -         -       -         -
dm mydg07       c1t99d0      -         17678493    -         -       -         -
dm mydg08       c1t100d0     -         17678493    -         -       -         -
```

The following commands would also achieve the same result:

```
# vxdg move mydg rootdg mydg01 mydg05
# vxdg move mydg rootdg vol1
```

Splitting disk groups

To remove a self-contained set of VxVM objects from an imported source disk group to a new target disk group, use the following command:

```
# vxdg [-o expand] [-o override|verify] split sourcedg targetdg \
    object ...
```

For a description of the `-o expand`, `-o override`, and `-o verify` options, see "Moving objects between disk groups" on page 203.

See "Splitting disk groups" on page 424 for more information on splitting shared disk groups in clusters.

For example, the following output from `vxprint` shows the contents of disk group `rootdg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC        KSTATE    LENGTH      PLOFFS    STATE   TUTIL0    PUTIL0
dg rootdg       rootdg       -         -           -         -       -         -
dm rootdg01     c0t1d0       -         17678493    -         -       -         -
dm rootdg02     c1t97d0      -         17678493    -         -       -         -
dm rootdg03     c1t112d0     -         1767849 3   -         -       -         -
dm rootdg04     c1t114d0     -         17678493    -         -       -         -
dm rootdg05     c1t96d0      -         17678493    -         -       -         -
dm rootdg06     c1t98d0      -         17678493    -         -       -         -
dm rootdg07     c1t99d0      -         17678493    -         -       -         -
dm rootdg08     c1t100d0     -         17678493    -         -       -         -
v  vol1         fsgen        ENABLED   2048        -         ACTIVE  -         -
pl vol1-01      vol1         ENABLED   3591        -         ACTIVE  -         -
sd rootdg01-01      vol1-01  ENABLED   3591        0         -       -         -
pl vol1-02      vol1         ENABLED   3591        -         ACTIVE  -         -
sd rootdg05-01      vol1-02  ENABLED   3591        0         -       -         -
```

The following command removes disks `rootdg07` and `rootdg08` from `rootdg` to form a new disk group, `mydg`:

```
# vxdg -o expand split rootdg mydg rootdg07 rootdg08
```

The moved volumes are initially disabled following the split. Use the following commands to recover and restart the volumes in the new target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the split shows the new disk group, `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC        KSTATE    LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg rootdg       rootdg       -         -          -         -       -         -
dm rootdg01     c0t1d0       -         17678493   -         -       -         -
dm rootdg02     c1t97d0      -         17678493   -         -       -         -
dm rootdg03     c1t112d0     -         1767849 3  -         -       -         -
dm rootdg04     c1t114d0     -         17678493   -         -       -         -
dm rootdg05     c1t96d0      -         17678493   -         -       -         -
dm rootdg06     c1t98d0      -         17678493   -         -       -         -
v  vol1         fsgen        ENABLED   2048       -         ACTIVE  -         -
pl vol1-01      vol1         ENABLED   3591       -         ACTIVE  -         -
sd rootdg01-01      vol1-01  ENABLED   3591       0         -       -         -
pl vol1-02      vol1         ENABLED   3591       -         ACTIVE  -         -
sd rootdg05-01      vol1-02  ENABLED   3591       0         -       -         -

Disk group: mydg
TY NAME         ASSOC        KSTATE    LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg mydg         mydg         -         -          -         -       -         -
dm rootdg07     c1t99d0      -         17678493   -         -       -         -
dm rootdg08     c1t100d0     -         17678493   -         -       -         -
```

Joining disk groups

To remove all VxVM objects from an imported source disk group to an imported target disk group, use the following command:

```
# vxdg [-o override|verify] join sourcedg targetdg
```

Note: You cannot specify `rootdg` as the source disk group for a `join` operation.

For a description of the `-o override` and `-o verify` options, see "Moving objects between disk groups" on page 203.

See "Joining disk groups" on page 424 for information on joining disk groups in a cluster.

For example, the following output from `vxprint` shows the contents of the disk group `rootdg` and `mydg`:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC        KSTATE    LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg rootdg       rootdg       -         -          -         -       -         -
dm rootdg01     c0t1d0       -         17678493   -         -       -         -
dm rootdg02     c1t97d0      -         17678493   -         -       -         -
dm rootdg03     c1t112d0     -         1767849 3  -         -       -         -
dm rootdg04     c1t114d0     -         17678493   -         -       -         -
dm rootdg07     c1t99d0      -         17678493   -         -       -         -
dm rootdg08     c1t100d0     -         17678493   -         -       -         -
```

```
Disk group: mydg
TY NAME         ASSOC        KSTATE   LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg mydg         mydg         -        -          -         -       -         -
dm mydg05       c1t96d0      -        17678493   -         -       -         -
dm mydg06       c1t98d0      -        17678493   -         -       -         -
v  vol1         fsgen        ENABLED  2048       -         ACTIVE  -         -
pl vol1-01      vol1         ENABLED  3591       -         ACTIVE  -         -
sd mydg01-01    vol1-01      ENABLED  3591       0         -       -         -
pl vol1-02      vol1         ENABLED  3591       -         ACTIVE  -         -
sd mydg05-01    vol1-02      ENABLED  3591       0         -       -         -
```

The following command joins disk group `mydg` to `rootdg`:

```
# vxdg join mydg rootdg
```

The moved volumes are initially disabled following the join. Use the following commands to recover and restart the volumes in the target disk group:

```
# vxrecover -g targetdg -m [volume ...]
# vxvol -g targetdg startall
```

The output from `vxprint` after the join shows that disk group `mydg` has been removed:

```
# vxprint
Disk group: rootdg
TY NAME         ASSOC        KSTATE   LENGTH     PLOFFS    STATE   TUTIL0    PUTIL0
dg rootdg       rootdg       -        -          -         -       -         -
dm mydg01       c0t1d0       -        17678493   -         -       -         -
dm rootdg02     c1t97d0      -        17678493   -         -       -         -
dm rootdg03     c1t112d0     -        1767849 3  -         -       -         -
dm rootdg04     c1t114d0     -        17678493   -         -       -         -
dm mydg05       c1t96d0      -        17678493   -         -       -         -
dm rootdg06     c1t98d0      -        17678493   -         -       -         -
dm rootdg07     c1t99d0      -        17678493   -         -       -         -
dm rootdg08     c1t100d0     -        17678493   -         -       -         -
v  vol1         fsgen        ENABLED  2048       -         ACTIVE  -         -
pl vol1-01      vol1         ENABLED  3591       -         ACTIVE  -         -
sd mydg01-01    vol1-01      ENABLED  3591       0         -       -         -
pl vol1-02      vol1         ENABLED  3591       -         ACTIVE  -         -
sd mydg05-01    vol1-02      ENABLED  3591       0         -       -         -
```

Disabling a disk group

To disable a disk group, unmount and stop any volumes in the disk group, and then use the following command to deport it:

```
# vxdg deport diskgroup
```

Deporting a disk group does not actually remove the disk group. It disables use of the disk group by the system. Disks in a deported disk group can be reused, reinitialized, added to other disk groups, or imported for use on other systems. Use the `vxdg import` command to re-enable access to the disk group.

Destroying a disk group

The vxdg command provides a destroy option that removes a disk group from the system and frees the disks in that disk group for reinitialization:

```
# vxdg destroy diskgroup
```

Caution: This command destroys all data on the disks.

When a disk group is destroyed, the disks that are released can be re-used in other disk groups.

Recovering a destroyed disk group

If a disk group has been accidentally destroyed, you can recover it, provided that the disks that were in the disk group have not been modified or reused elsewhere.

To recover a destroyed disk group

1. Enter the following command to find out the disk group ID (*dgid*) of one of the disks that was in the disk group:

   ```
   # vxdisk -s list disk_access_name
   ```

 The disk must be specified by its disk access name, such as c0t12d0. Examine the output from the command for a line similar to the following that specifies the disk group ID.

   ```
   dgid:   963504895.1075.bass
   ```

2. Use the disk group ID to import the disk group:

   ```
   # vxdg import dgid
   ```

Upgrading a disk group

Note: On some platforms, the first release of Veritas Volume Manager was 3.0 or 3.2.

Prior to the release of Veritas Volume Manager 3.0, the disk group version was automatically upgraded (if needed) when the disk group was imported.

From release 3.0 of Veritas Volume Manager, the two operations of importing a disk group and upgrading its version are separate. You can import a disk group from a previous version and use it without upgrading it.

When you want to use new features, the disk group can be upgraded. The upgrade is an explicit operation. Once the upgrade occurs, the disk group

becomes incompatible with earlier releases of VxVM that do not support the new version.

Before the imported disk group is upgraded, no changes are made to the disk group to prevent its use on the release from which it was imported until you explicitly upgrade it to the current release.

Until completion of the upgrade, the disk group can be used "as is" provided there is no attempt to use the features of the current version. Attempts to use a feature of the current version that is not a feature of the version from which the disk group was imported results in an error message similar to this:

```
VxVM vxedit ERROR V-5-1-2829 Disk group version doesn't support
feature
```

To use any of the new features, you must run the `vxdg upgrade` command to explicitly upgrade the disk group to a version that supports those features.

All disk groups have a version number associated with them. Veritas Volume Manager releases support a specific set of disk group versions. VxVM can import and perform operations on a disk group of that version. The operations are limited by what features and operations the disk group version supports.

The table, "Disk group version assignments," summarizes the Veritas Volume Manager releases that introduce and support specific disk group versions:

Table 4-1 Disk group version assignments

VxVM release	Introduces disk group version	Supports disk group versions
1.2	10	10
1.3	15	15
2.0	20	20
2.2	30	30
2.3	40	40
2.5	50	50
3.0	60	20-40, 60
3.1	70	20-70
3.1.1	80	20-80
3.2, 3.5	90	20-90
4.0	110	20-110
4.1	120	20-120
5.0	140	20-140

Importing the disk group of a previous version on a Veritas Volume Manager system prevents the use of features introduced since that version was released. The table, "Features supported by disk group versions," summarizes the features that are supported by disk group versions 20 through 140:

Table 4-2 Features supported by disk group versions

Disk group version	New features supported	Previous version features supported
140	■ Data Migration Features ■ DMP Enhancements ■ Import of Cloned Disks ■ Intelligent Storage Provisioning (ISP) Enhancements ■ Remote Mirror (Campus Cluster) ■ Veritas Volume Replicator (VVR) Enhancements	20, 30, 40, 50, 60, 70, 80, 90, 110, 120, 130
130	■ VVR Enhancements	20, 30, 40, 50, 60, 70, 80, 90, 110, 120
120	■ Automatic Cluster-wide Failback for A/P arrays ■ Migration of Volumes to ISP ■ Persistent DMP Policies ■ Shared Disk Group Failure Policy	20, 30, 40, 50, 60, 70, 80, 90, 110
110	■ Cross-platform Data Sharing (CDS) ■ Device Discovery Layer (DDL) 2.0 ■ Disk Group Configuration Backup and Restore ■ Elimination of `rootdg` as a Special Disk Group ■ Full-Sized and Space-Optimized Instant Snapshots ■ Intelligent Storage Provisioning (ISP) ■ Serial Split Brain Detection ■ Volume Sets (Multiple Device Support for VxFS)	20, 30, 40, 50, 60, 70, 80, 90
90	■ Cluster Support for Oracle Resilvering ■ Disk Group Move, Split and Join ■ Device Discovery Layer (DDL) 1.0 ■ Layered Volume Support in Clusters ■ Ordered Allocation ■ OS Independent Naming Support ■ Persistent FastResync	20, 30, 40, 50, 60, 70, 80
80	■ VVR Enhancements	20, 30, 40, 50, 60, 70

Table 4-2 Features supported by disk group versions

Disk group version	New features supported	Previous version features supported
70	■ Non-Persistent FastResync ■ Sequential DRL ■ Unrelocate ■ VVR Enhancements	20, 30, 40, 50, 60
60	■ Online Relayout ■ Safe RAID-5 Subdisk Moves	20, 30, 40
50	■ SRVM (now known as Veritas Volume Replicator or VVR)	20, 30, 40
40	■ Hot-Relocation	20, 30
30	■ VxSmartSync Recovery Accelerator	20
20	■ Dirty Region Logging (DRL) ■ Disk Group Configuration Copy Limiting ■ Mirrored Volumes Logging ■ New-Style Stripes ■ RAID-5 Volumes ■ Recovery Checkpointing	

To list the version of a disk group, use this command:

```
# vxdg list dgname
```

You can also determine the disk group version by using the `vxprint` command with the `-l` format option.

To upgrade a disk group to the highest version supported by the release of VxVM that is currently running, use this command:

```
# vxdg upgrade dgname
```

By default, VxVM creates a disk group of the highest version supported by the release. For example, Veritas Volume Manager 5.0 creates disk groups with version 140.

It may sometimes be necessary to create a disk group for an older version. The default disk group version for a disk group created on a system running Veritas Volume Manager 5.0 is 140. Such a disk group cannot be imported on a system running Veritas Volume Manager 4.1, as that release only supports up to version 120. Therefore, to create a disk group on a system running Veritas Volume Manager 5.0 that can be imported by a system running Veritas Volume Manager 4.1, the disk group must be created with a version of 120 or less.

To create a disk group with a previous version, specify the `-T` *version* option to the `vxdg init` command.

For example, to create a disk group with version 120 that can be imported by a system running VxVM 4.1, use the following command:

```
# vxdg -T 120 init newdg newdg01=c0t3d0
```

This creates a disk group, `newdg`, which can be imported by Veritas Volume Manager 4.1. Note that while this disk group can be imported on the VxVM 4.1 system, attempts to use features from Veritas Volume Manager 5.0 will fail.

Managing the configuration daemon in VxVM

The VxVM configuration daemon (`vxconfigd`) provides the interface between VxVM commands and the kernel device drivers. `vxconfigd` handles configuration change requests from VxVM utilities, communicates the change requests to the VxVM kernel, and modifies configuration information stored on disk. `vxconfigd` also initializes VxVM when the system is booted.

The `vxdctl` command is the command-line interface to the `vxconfigd` daemon.

You can use `vxdctl` to:

- Control the operation of the `vxconfigd` daemon.
- Change the system-wide definition of the default disk group.

Note: In VxVM 4.0 and later releases, disk access records are no longer stored in the `/etc/vx/volboot` file. Non-persistent disk access records are created by scanning the disks at system startup. Persistent disk access records for `simple` and `nopriv` disks are permanently stored in the `/etc/vx/darecs` file in the `root` file system. The `vxconfigd` daemon reads the contents of this file to locate the disks and the configuration databases for their disk groups. (The `/etc/vx/darecs` file is also used to store definitions of foreign devices that are not autoconfigurable. Such entries may be added by using the `vxddladm addforeign` command. See the `vxddladm`(1M) manual page for more information.)

You can also use `vxdctl` to:

- Reconfigure the DMP database to include disk devices newly attached to, or removed from the system.
- Create DMP device nodes in the directories /`dev/vx/dmp` and `/dev/vx/rdmp`.
- Update the DMP database with changes in path type for active/passive disk arrays. Use the utilities provided by the disk-array vendor to change the path type between primary and secondary.

For more information about how to use vxdctl, refer to the vxdctl(1M) manual page.

Backing up and restoring disk group configuration data

The disk group configuration backup and restoration feature allows you to back up and restore all configuration data for disk groups, and for VxVM objects such as volumes that are configured within the disk groups. The vxconfigbackupd daemon monitors changes to the VxVM configuration and automatically records any configuration changes that occur. Two utilities, vxconfigbackup and vxconfigrestore, are provided for backing up and restoring a VxVM configuration for a disk group.

For information on backing up and restoring disk group configurations, see the "Backing Up and Restoring Disk Group Configurations" chapter in the *Veritas Volume Manager Troubleshooting Guide*, and the vxconfigbackup(1M) and vxconfigrestore(1M) manual pages.

Using vxnotify to monitor configuration changes

You can use the vxnotify utility to display events relating to disk and configuration changes that are managed by the vxconfigd configuration daemon. If vxnotify is running on a system where the VxVM clustering feature is active, it displays events that are related to changes in the cluster state of the system on which it is running. The vxnotify utility displays the requested event types until you kill it, until it has received a specified number of events, or until a specified period of time has elapsed.

Examples of configuration events that can be detected include disabling and enabling of controllers, paths and DMP nodes, RAID-5 volumes entering degraded mode, detachment of disks, plexes and volumes, and nodes joining and leaving a cluster.

For example, the following vxnotify command displays information about all disk, plex, and volume detachments as they occur:

```
# vxnotify -f
```

The following command provides information about cluster configuration changes, including the import and deport of shared disk groups:

```
# vxnotify -s -i
```

For more information about the vxnotify utility, and the types of configuration events that it can report, see the vxnotify(1M) manual page.

Chapter 5

Creating and administering subdisks

This chapter describes how to create and maintain *subdisks*. Subdisks are the low-level building blocks in a Veritas Volume Mananger (VxVM) configuration that are required to create plexes and volumes.

Note: Most VxVM commands require superuser or equivalent privileges.

Creating subdisks

Note: Subdisks are created automatically if you use the `vxassist` command or the Veritas Enterprise Administrator (VEA) to create volumes. For more information, see "Creating a volume" on page 238.

Use the `vxmake` command to create VxVM objects, such as subdisks:

```
# vxmake [-g diskgroup] sd subdisk diskname,offset,length
```

where: *subdisk* is the name of the subdisk, *diskname* is the disk name, *offset* is the starting point (offset) of the subdisk within the disk, and *length* is the length of the subdisk.

For example, to create a subdisk named `mydg02-01` in the disk group, `mydg`, that starts at the beginning of disk `mydg02` and has a length of 8000 sectors, use the following command:

```
# vxmake -g mydg sd mydg02-01 mydg02,0,8000
```

Note: As for all VxVM commands, the default size unit is `s`, representing a sector. Add a suffix, such as `k` for kilobyte, `m` for megabyte or `g` for gigabyte, to change the unit of size. For example, `500m` would represent 500 megabytes.

If you intend to use the new subdisk to build a volume, you must associate the subdisk with a plex (see "Associating subdisks with plexes" on page 218). Subdisks for all plex layouts (concatenated, striped, RAID-5) are created the same way.

Displaying subdisk information

The `vxprint` command displays information about VxVM objects. To display general information for all subdisks, use this command:

```
# vxprint -st
```

The `-s` option specifies information about subdisks. The `-t` option prints a single-line output record that depends on the type of object being listed.

The following is example output:

```
SD NAME      PLEX     DISK    DISKOFFS LENGTH [COL/]OFF DEVICE   MODE
SV NAME      PLEX     VOLNAME NVOLLAYR LENGTH [COL/]OFF AM/NM    MODE
sd mydg01-01 vol1-01  mydg01  0        102400 0       c2t0d1  ENA
sd mydg02-01 vol2-01  mydg02  0        102400 0       c2t1d1  ENA
```

You can display complete information about a particular subdisk by using this command:

```
# vxprint [-g diskgroup] -l subdisk
```

For example, the following command displays all information for subdisk `mydg02-01` in the disk group, `mydg`:

```
# vxprint -g mydg -l mydg02-01
```

This command provides the following output:

```
Disk group: mydg

Subdisk:  mydg02-01
info:     disk=mydg02 offset=0 len=205632
assoc:    vol=mvol plex=mvol-02 (offset=0)
flags:    enabled
device:   device=c2t1d1 path=/dev/vx/dmp/c2t1d1 diskdev=32/68
```

Moving subdisks

Moving a subdisk copies the disk space contents of a subdisk onto one or more other subdisks. If the subdisk being moved is associated with a plex, then the data stored on the original subdisk is copied to the new subdisks. The old subdisk is dissociated from the plex, and the new subdisks are associated with the plex. The association is at the same offset within the plex as the source subdisk. To move a subdisk, use the following command:

```
# vxsd [-g diskgroup] mv old_subdisk new_subdisk [new_subdisk ...]
```

For example, if `mydg03` in the disk group, `mydg`, is to be evacuated, and `mydg12` has enough room on two of its subdisks, use the following command:

```
# vxsd -g mydg mv mydg03-01 mydg12-01 mydg12-02
```

For the subdisk move to work correctly, the following conditions must be met:

- The subdisks involved must be the same size.
- The subdisk being moved must be part of an active plex on an active (`ENABLED`) volume.
- The new subdisk must not be associated with any other plex.

See "Configuring hot-relocation to use only spare disks" on page 390 for information about manually relocating subdisks after hot-relocation.

Splitting subdisks

Splitting a subdisk divides an existing subdisk into two separate subdisks. To split a subdisk, use the following command:

```
# vxsd [-g diskgroup] -s size split subdisk newsd1 newsd2
```

where *subdisk* is the name of the original subdisk, *newsd1* is the name of the first of the two subdisks to be created and *newsd2* is the name of the second subdisk to be created.

The `-s` option is required to specify the size of the *first* of the two subdisks to be created. The second subdisk occupies the remaining space used by the original subdisk.

If the original subdisk is associated with a plex before the task, upon completion of the split, both of the resulting subdisks are associated with the same plex.

To split the original subdisk into more than two subdisks, repeat the previous command as many times as necessary on the resulting subdisks.

For example, to split subdisk mydg03-02, with size 2000 megabytes into subdisks mydg03-02, mydg03-03, mydg03-04 and mydg03-05, each with size 500 megabytes, all in the disk group, mydg, use the following commands:

```
# vxsd -g mydg -s 1000m split mydg03-02 mydg03-02 mydg03-04
# vxsd -g mydg -s 500m split mydg03-02 mydg03-02 mydg03-03
# vxsd -g mydg -s 500m split mydg03-04 mydg03-04 mydg03-05
```

Joining subdisks

Joining subdisks combines two or more existing subdisks into one subdisk. To join subdisks, the subdisks must be contiguous on the same disk. If the selected subdisks are associated, they must be associated with the same plex, and be contiguous in that plex. To join several subdisks, use the following command:

```
# vxsd [-g diskgroup] join subdisk1 subdisk2 ... new_subdisk
```

For example, to join the contiguous subdisks mydg03-02, mydg03-03, mydg03-04 and mydg03-05 as subdisk mydg03-02 in the disk group, mydg, use the following command:

```
# vxsd -g mydg join mydg03-02 mydg03-03 mydg03-04 mydg03-05 \
  mydg03-02
```

Associating subdisks with plexes

Associating a subdisk with a plex places the amount of disk space defined by the subdisk at a specific offset within the plex. The entire area that the subdisk fills must not be occupied by any portion of another subdisk. There are several ways that subdisks can be associated with plexes, depending on the overall state of the configuration.

If you have already created all the subdisks needed for a particular plex, to associate subdisks at plex creation, use the following command:

```
# vxmake [-g diskgroup] plex plex sd=subdisk,...
```

For example, to create the plex home-1 and associate subdisks mydg02-01, mydg02-00, and mydg02-02 with plex home-1, all in the disk group, mydg, use the following command:

```
# vxmake -g mydg plex home-1 sd=mydg02-01,mydg02-00,mydg02-02
```

Subdisks are associated in order starting at offset 0. If you use this type of command, you do not have to specify the multiple commands needed to create the plex and then associate each of the subdisks with that plex. In this example, the subdisks are associated to the plex in the order they are listed (after sd=). The disk space defined as mydg02-01 is first, mydg02-00 is second, and mydg02-02 is third. This method of associating subdisks is convenient during initial configuration.

Subdisks can also be associated with a plex that already exists. To associate one or more subdisks with an existing plex, use the following command:

```
# vxsd [-g diskgroup] assoc plex subdisk1 [subdisk2 subdisk3 ...]
```

For example, to associate subdisks named `mydg02-01`, `mydg02-00`, and `mydg02-02` with a plex named `home-1`, use the following command:

```
# vxsd -g mydg assoc home-1 mydg02-01 mydg02-00 mydg02-01
```

If the plex is not empty, the new subdisks are added after any subdisks that are already associated with the plex, unless the `-l` option is specified with the command. The `-l` option associates subdisks at a specific offset within the plex.

The `-l` option is required if you previously created a sparse plex (that is, a plex with portions of its address space that do not map to subdisks) for a particular volume, and subsequently want to make the plex complete. To complete the plex, create a subdisk of a size that fits the hole in the sparse plex exactly. Then, associate the subdisk with the plex by specifying the offset of the beginning of the hole in the plex, using the following command:

```
# vxsd [-g diskgroup] -l offset assoc sparse_plex exact_size_subdisk
```

For example, the following command would insert the subdisk, `mydg15-01`, in the plex, `vol10-01`, starting at an offset of 4096 blocks:

```
# vxsd -g mydg -l 4096b assoc vol10-01 mydg15-01
```

Note: The subdisk must be exactly the right size. VxVM does not allow the space defined for two subdisks to overlap within a plex.

For striped or RAID-5 plexes, use the following command to specify a column number and column offset for the subdisk to be added:

```
# vxsd [-g diskgroup] -l column_#/offset assoc plex subdisk ...
```

If only one number is specified with the `-l` option for striped plexes, the number is interpreted as a column number and the subdisk is associated at the end of the column.

For example, the following command would add the subdisk, `mydg11-01`, to the end of column 1 of the plex, `vol02-01`:

```
# vxsd -g mydg -l 1 assoc vol02-01 mydg11-01
```

Alternatively, to add *M* subdisks at the end of each of the *N* columns in a striped or RAID-5 volume, you can use the following form of the `vxsd` command:

```
# vxsd [-g diskgroup] assoc plex subdisk1:0 ... subdiskM:N-1
```

The following example shows how to append three subdisk to the ends of the three columns in a striped plex, `vol-01`, in the disk group, `mydg`:

```
# vxsd -g mydg assoc vol01-01 mydg10-01:0 mydg11-01:1 \
  mydg12-01:2
```

If a subdisk is filling a “hole” in the plex (that is, some portion of the volume logical address space is mapped by the subdisk), the subdisk is considered stale.

If the volume is enabled, the association operation regenerates data that belongs on the subdisk. Otherwise, it is marked as stale and is recovered when the volume is started.

Associating log subdisks

Note: The version 20 DCO volume layout includes space for a DRL. Do not apply the procedure described in this section to a volume that has a version 20 DCO volume associated with it. See "Preparing a volume for DRL and instant snapshots" on page 275 for more information.

Log subdisks are defined and added to a plex that is to become part of a volume on which dirty region logging (DRL) is enabled. DRL is enabled for a volume when the volume is mirrored and has at least one log subdisk.

For a description of DRL, see "Dirty region logging" on page 60. Log subdisks are ignored as far as the usual plex policies are concerned, and are only used to hold the dirty region log.

Note: Only one log subdisk can be associated with a plex. Because this log subdisk is frequently written, care should be taken to position it on a disk that is not heavily used. Placing a log subdisk on a heavily-used disk can degrade system performance.

To add a log subdisk to an existing plex, use the following command:

```
# vxsd [-g diskgroup] aslog plex subdisk
```

where *subdisk* is the name to be used for the log subdisk. The plex must be associated with a mirrored volume before dirty region logging takes effect.

For example, to associate a subdisk named `mydg02-01` with a plex named `vol01-02`, which is already associated with volume `vol01` in the disk group, `mydg`, use the following command:

```
# vxsd -g mydg aslog vol01-02 mydg02-01
```

You can also add a log subdisk to an existing volume with the following command:

```
# vxassist [-g diskgroup] addlog volume disk
```

This command automatically creates a log subdisk within a log plex on the specified disk for the specified volume.

Dissociating subdisks from plexes

To break an established connection between a subdisk and the plex to which it belongs, the subdisk is *dissociated* from the plex. A subdisk is dissociated when the subdisk is removed or used in another plex. To dissociate a subdisk, use the following command:

```
# vxsd [-g diskgroup] [-o force] dis subdisk
```

For example, to dissociate a subdisk named `mydg02-01` from the plex with which it is currently associated in the disk group, `mydg`, use the following command:

```
# vxsd -g mydg dis mydg02-01
```

You can additionally remove the dissociated subdisks from VxVM control using the following form of the command:

```
# vxsd [-g diskgroup] -o rm dis subdisk
```

Caution: If the subdisk maps a portion of a volume's address space, dissociating it places the volume in DEGRADED mode. In this case, the `dis` operation prints a warning and must be forced using the `-o force` option to succeed. Also, if removing the subdisk makes the volume unusable, because another subdisk in the same stripe is unusable or missing and the volume is not DISABLED and empty, the operation is not allowed.

Removing subdisks

To remove a subdisk, use the following command:

```
# vxedit [-g diskgroup] rm subdisk
```

For example, to remove a subdisk named `mydg02-01` from the disk group, `mydg`, use the following command:

```
# vxedit -g mydg rm mydg02-01
```

Changing subdisk attributes

Caution: Change subdisk attributes with extreme care.

The `vxedit` command changes attributes of subdisks and other VxVM objects. To change subdisk attributes, use the following command:

```
# vxedit [-g diskgroup] set attribute=value ... subdisk ...
```

Subdisk fields that can be changed using the `vxedit` command include:

- `name`

- putil*n*
- tutil*n*
- len
- comment

The putil*n* field attributes are maintained on reboot; tutil*n* fields are temporary and are not retained on reboot. VxVM sets the putil0 and tutil0 utility fields. Other Symantec products, such as the Veritas Enterprise Administrator (VEA), set the putil1 and tutil1 fields. The putil2 and tutil2 are available for you to use for site-specific purposes. The length field, len, can only be changed if the subdisk is dissociated.

For example, to change the comment field of a subdisk named mydg02-01 in the disk group, mydg, use the following command:

```
# vxedit -g mydg set comment="subdisk comment" mydg02-01
```

To prevent a particular subdisk from being associated with a plex, set the putil0 field to a non-null string, as shown in the following command:

```
# vxedit -g mydg set putil0="DO-NOT-USE" mydg02-01
```

See the vxedit(1M) manual page for more information about using the vxedit command to change the attribute fields of VxVM objects.

Chapter 6

Creating and administering plexes

This chapter describes how to create and maintain *plexes*. Plexes are logical groupings of subdisks that create an area of disk space independent of physical disk size or other restrictions. Replication (mirroring) of disk data is set up by creating multiple data plexes for a single volume. Each data plex in a mirrored volume contains an identical copy of the volume data. Because each data plex must reside on different disks from the other plexes, the replication provided by mirroring prevents data loss in the event of a single-point disk-subsystem failure. Multiple data plexes also provide increased data integrity and reliability.

Note: Most VxVM commands require superuser or equivalent privileges.

Creating plexes

Note: Plexes are created automatically if you use the `vxassist` command or the Veritas Enterprise Administrator (VEA) to create volumes. For more information, see "Creating a volume" on page 238.

Use the `vxmake` command to create VxVM objects, such as plexes. When creating a plex, identify the subdisks that are to be associated with it:

To create a plex from existing subdisks, use the following command:

```
# vxmake [-g diskgroup] plex plex sd=subdisk1[,subdisk2,...]
```

For example, to create a concatenated plex named `vol01-02` from two existing subdisks named `mydg02-01` and `mydg02-02` in the disk group, `mydg`, use the following command:

```
# vxmake -g mydg plex vol01-02 sd=mydg02-01,mydg02-02
```

Creating a striped plex

To create a striped plex, you must specify additional attributes. For example, to create a striped plex named `pl-01` in the disk group, `mydg`, with a stripe width of 32 sectors and 2 columns, use the following command:

```
# vxmake -g mydg plex pl-01 layout=stripe stwidth=32 ncolumn=2 \
  sd=mydg01-01,mydg02-01
```

To use a plex to build a volume, you must associate the plex with the volume. For more information, see the section, "Attaching and associating plexes" on page 229.

Displaying plex information

Listing plexes helps identify free plexes for building volumes. Use the plex (`-p`) option to the `vxprint` command to list information about all plexes.

To display detailed information about all plexes in the system, use the following command:

```
# vxprint -lp
```

To display detailed information about a specific plex, use the following command:

```
# vxprint [-g diskgroup] -l plex
```

The `-t` option prints a single line of information about the plex. To list free plexes, use the following command:

```
# vxprint -pt
```

The following section describes the meaning of the various plex states that may be displayed in the STATE field of `vxprint` output.

Plex states

Plex states reflect whether or not plexes are complete and are consistent copies (mirrors) of the volume contents. VxVM utilities automatically maintain the plex state. However, if a volume should not be written to because there are changes to that volume and if a plex is associated with that volume, you can modify the state of the plex. For example, if a disk with a particular plex located on it begins to fail, you can temporarily disable that plex.

Note: A plex does not have to be associated with a volume. A plex can be created with the `vxmake plex` command and be attached to a volume later.

VxVM utilities use plex states to:

- indicate whether volume contents have been initialized to a known state
- determine if a plex contains a valid copy (mirror) of the volume contents
- track whether a plex was in active use at the time of a system failure
- monitor operations on plexes

This section explains the individual plex states in detail. For more information about the possible transitions between plex states and how these are applied during volume recovery, see the chapter “Understanding the Plex State Cycle” in the section “Recovery from Hardware Failure” in the *Veritas Volume Manager Troubleshooting Guide*.

Plexes that are associated with a volume have one of the following states:

ACTIVE plex state

A plex can be in the ACTIVE state in two ways:

- when the volume is started and the plex fully participates in normal volume I/O (the plex contents change as the contents of the volume change)
- when the volume is stopped as a result of a system crash and the plex is ACTIVE at the moment of the crash

In the latter case, a system failure can leave plex contents in an inconsistent state. When a volume is started, VxVM does the recovery action to guarantee that the contents of the plexes marked as ACTIVE are made identical.

Note: On a system running well, ACTIVE should be the most common state you see for any volume plexes.

CLEAN plex state

A plex is in a CLEAN state when it is known to contain a consistent copy (mirror) of the volume contents and an operation has disabled the volume. As a result, when all plexes of a volume are clean, no action is required to guarantee that the plexes are identical when that volume is started.

DCOSNP plex state

This state indicates that a data change object (DCO) plex attached to a volume can be used by a snapshot plex to create a DCO volume during a snapshot operation.

EMPTY plex state

Volume creation sets all plexes associated with the volume to the EMPTY state to indicate that the plex is not yet initialized.

IOFAIL plex state

The IOFAIL plex state is associated with persistent state logging. When the `vxconfigd` daemon detects an uncorrectable I/O failure on an ACTIVE plex, it places the plex in the IOFAIL state to exclude it from the recovery selection process at volume start time.

This state indicates that the plex is out-of-date with respect to the volume, and that it requires complete recovery. It is likely that one or more of the disks associated with the plex should be replaced.

LOG plex state

The state of a dirty region logging (DRL) or RAID-5 log plex is always set to LOG.

OFFLINE plex state

The `vxmend off` task indefinitely detaches a plex from a volume by setting the plex state to OFFLINE. Although the detached plex maintains its association with the volume, changes to the volume do not update the OFFLINE plex. The plex is not updated until the plex is put online and reattached with the `vxplex att` task. When this occurs, the plex is placed in the STALE state, which causes its contents to be recovered at the next `vxvol start` operation.

SNAPATT plex state

This state indicates a snapshot plex that is being attached by the snapstart operation. When the attach is complete, the state for the plex is changed to SNAPDONE. If the system fails before the attach completes, the plex and all of its subdisks are removed.

SNAPDIS plex state

This state indicates a snapshot plex that is fully attached. A plex in this state can be turned into a snapshot volume with the `vxplex snapshot` command. If the system fails before the attach completes, the plex is dissociated from the volume. See the `vxplex`(1M) manual page for more information.

SNAPDONE plex state

The SNAPDONE plex state indicates that a snapshot plex is ready for a snapshot to be taken using `vxassist snapshot`.

SNAPTMP plex state

The SNAPTMP plex state is used during a `vxassist snapstart` operation when a snapshot is being prepared on a volume.

STALE plex state

If there is a possibility that a plex does not have the complete and current volume contents, that plex is placed in the STALE state. Also, if an I/O error occurs on a plex, the kernel stops using and updating the contents of that plex, and the plex state is set to STALE.

A `vxplex att` operation recovers the contents of a STALE plex from an ACTIVE plex. Atomic copy operations copy the contents of the volume to the STALE plexes. The system administrator can force a plex to the STALE state with a `vxplex det` operation.

TEMP plex state

Setting a plex to the TEMP state eases some plex operations that cannot occur in a truly atomic fashion. For example, attaching a plex to an enabled volume requires copying volume contents to the plex before it can be considered fully attached.

A utility sets the plex state to TEMP at the start of such an operation and to an appropriate state at the end of the operation. If the system fails for any reason, a TEMP plex state indicates that the operation is incomplete. A later `vxvol start` dissociates plexes in the TEMP state.

TEMPRM plex state

A TEMPRM plex state is similar to a TEMP state except that at the completion of the operation, the TEMPRM plex is removed. Some subdisk operations require a temporary plex. Associating a subdisk with a plex, for example, requires updating the subdisk with the volume contents before actually associating the subdisk. This update requires associating the subdisk with a temporary plex, marked TEMPRM, until the operation completes and removes the TEMPRM plex.

If the system fails for any reason, the TEMPRM state indicates that the operation did not complete successfully. A later operation dissociates and removes TEMPRM plexes.

TEMPRMSD plex state

The TEMPRMSD plex state is used by `vxassist` when attaching new data plexes to a volume. If the synchronization operation does not complete, the plex and its subdisks are removed.

Plex condition flags

`vxprint` may also display one of the following condition flags in the STATE field:

IOFAIL plex condition

The plex was detached as a result of an I/O failure detected during normal volume I/O. The plex is out-of-date with respect to the volume, and in need of complete recovery. However, this condition also indicates a likelihood that one of the disks in the system should be replaced.

NODAREC plex condition

No physical disk was found for one of the subdisks in the plex. This implies either that the physical disk failed, making it unrecognizable, or that the physical disk is no longer attached through a known access path. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

NODEVICE plex condition

A physical device could not be found corresponding to the disk ID in the disk media record for one of the subdisks associated with the plex. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

RECOVER plex condition

A disk corresponding to one of the disk media records was replaced, or was reattached too late to prevent the plex from becoming out-of-date with respect to the volume. The plex required complete recovery from another plex in the volume to synchronize its contents.

REMOVED plex condition

Set in the disk media record when one of the subdisks associated with the plex is removed. The plex cannot be used until this condition is fixed, or the affected subdisk is dissociated.

Plex kernel states

The *plex kernel state* indicates the accessibility of the plex to the volume driver which monitors it.

Note: No user intervention is required to set these states; they are maintained internally. On a system that is operating properly, all plexes are enabled.

The following plex kernel states are defined:

DETACHED plex kernel state

Maintenance is being performed on the plex. Any write request to the volume is not reflected in the plex. A read request from the volume is not satisfied from the plex. Plex operations and `ioctl` function calls are accepted.

DISABLED plex kernel state

The plex is offline and cannot be accessed.

ENABLED plex kernel state

The plex is online. A write request to the volume is reflected in the plex. A read request from the volume is satisfied from the plex. If a plex is sparse, this is indicated by the `SPARSE` modifier being displayed in the output from the `vxprint -t` command.

Attaching and associating plexes

A plex becomes a participating plex for a volume by attaching it to a volume. (Attaching a plex associates it with the volume and enables the plex for use.) To attach a plex to an existing volume, use the following command:

```
# vxplex [-g diskgroup] att volume plex
```

For example, to attach a plex named `vol01-02` to a volume named `vol01` in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg att vol01 vol01-02
```

If the volume does not already exist, a plex (or multiple plexes) can be associated with the volume when it is created using the following command:

```
# vxmake [-g diskgroup] -U usetype vol volume plex=plex1[,plex2...]
```

For example, to create a mirrored, `fsgen`-type volume named `home`, and to associate two existing plexes named `home-1` and `home-2` with `home`, use the following command:

```
# vxmake -g mydg -U fsgen vol home plex=home-1,home-2
```

Note: You can also use the command **vxassist mirror *volume*** to add a data plex as a mirror to an existing volume.

Taking plexes offline

Once a volume has been created and placed online (ENABLED), VxVM can temporarily disconnect plexes from the volume. This is useful, for example, when the hardware on which the plex resides needs repair or when a volume has been left unstartable and a source plex for the volume revive must be chosen manually.

Resolving a disk or system failure includes taking a volume offline and attaching and detaching its plexes. The two commands used to accomplish disk failure resolution are vxmend and vxplex.

To take a plex OFFLINE so that repair or maintenance can be performed on the physical disk containing subdisks of that plex, use the following command:

```
# vxmend [-g diskgroup] off plex
```

If a disk fails (for example, it has a head crash), you can use the vxmend command to take offline all plexes that have associated subdisks on the affected disk. For example, if plexes vol01-02 and vol02-02 in the disk group, mydg, had subdisks on a drive to be repaired, use the following command to take these plexes offline:

```
# vxmend -g mydg off vol01-02 vol02-02
```

This command places vol01-02 and vol02-02 in the OFFLINE state, and they remain in that state until it is changed. The plexes are not automatically recovered on rebooting the system.

Detaching plexes

To temporarily detach one data plex in a mirrored volume, use the following command:

```
# vxplex [-g diskgroup] det plex
```

For example, to temporarily detach a plex named vol01-02 in the disk group, mydg, and place it in maintenance mode, use the following command:

```
# vxplex -g mydg det vol01-02
```

This command temporarily detaches the plex, but maintains the association between the plex and its volume. However, the plex is not used for I/O. A plex detached with the preceding command is recovered at system reboot. The plex state is set to STALE, so that if a vxvol start command is run on the appropriate volume (for example, on system reboot), the contents of the plex is recovered and made ACTIVE.

When the plex is ready to return as an active part of its volume, follow the procedures in the following section, "Reattaching plexes."

Reattaching plexes

When a disk has been repaired or replaced and is again ready for use, the plexes must be put back online (plex state set to ACTIVE). To set the plexes to ACTIVE, use one of the following procedures depending on the state of the volume.

- If the volume is currently ENABLED, use the following command to reattach the plex:

  ```
  # vxplex [-g diskgroup] att volume plex ...
  ```

 For example, for a plex named vol01-02 on a volume named vol01 in the disk group, mydg, use the following command:

  ```
  # vxplex -g mydg att vol01 vol01-02
  ```

 As when returning an OFFLINE plex to ACTIVE, this command starts to recover the contents of the plex and, after the recovery is complete, sets the plex utility state to ACTIVE.

- If the volume is not in use (not ENABLED), use the following command to re-enable the plex for use:

  ```
  # vxmend [-g diskgroup] on plex
  ```

 For example, to re-enable a plex named vol01-02 in the disk group, mydg, enter:

  ```
  # vxmend -g mydg on vol01-02
  ```

 In this case, the state of vol01-02 is set to STALE. When the volume is next started, the data on the plex is revived from another plex, and incorporated into the volume with its state set to ACTIVE.

If the vxinfo command shows that the volume is unstartable (see "Listing Unstartable Volumes" in the section "Recovery from Hardware Failure" in the *Veritas Volume Manager Troubleshooting Guide*), set one of the plexes to CLEAN using the following command:

```
# vxmend [-g diskgroup] fix clean plex
```

Start the volume using the following command:

```
# vxvol [-g diskgroup] start volume
```

Moving plexes

Moving a plex copies the data content from the original plex onto a new plex. To move a plex, use the following command:

```
# vxplex [-g diskgroup] mv original_plex new_plex
```

For a move task to be successful, the following criteria must be met:

- The old plex must be an active part of an active (ENABLED) volume.
- The new plex must be at least the same size or larger than the old plex.
- The new plex must not be associated with another volume.

The size of the plex has several implications:

- If the new plex is smaller or more sparse than the original plex, an incomplete copy is made of the data on the original plex. If an incomplete copy is desired, use the -o force option to vxplex.
- If the new plex is longer or less sparse than the original plex, the data that exists on the original plex is copied onto the new plex. Any area that is not on the original plex, but is represented on the new plex, is filled from other complete plexes associated with the same volume.
- If the new plex is longer than the volume itself, then the remaining area of the new plex above the size of the volume is not initialized and remains unused.

Copying volumes to plexes

This task copies the contents of a volume onto a specified plex. The volume to be copied must not be enabled. The plex cannot be associated with any other volume. To copy a plex, use the following command:

```
# vxplex [-g diskgroup] cp volume new_plex
```

After the copy task is complete, *new_plex* is not associated with the specified volume *volume*. The plex contains a complete copy of the volume data. The plex that is being copied should be the same size or larger than the volume. If the plex being copied is larger than the volume, an incomplete copy of the data results. For the same reason, *new_plex* should not be sparse.

Dissociating and removing plexes

When a plex is no longer needed, you can dissociate it from its volume and remove it as an object from VxVM. You might want to remove a plex for the following reasons:

- to provide free disk space
- to reduce the number of mirrors in a volume so you can increase the length of another mirror and its associated volume. When the plexes and subdisks are removed, the resulting space can be added to other volumes
- to remove a temporary mirror that was created to back up a volume and is no longer needed
- to change the layout of a plex

Caution: To save the data on a plex to be removed, the configuration of that plex must be known. Parameters from that configuration (stripe unit size and subdisk ordering) are critical to the creation of a new plex to contain the same data. Before a plex is removed, you must record its configuration. See "Displaying plex information" on page 224" for more information.

To dissociate a plex from the associated volume and remove it as an object from VxVM, use the following command:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove a plex named `vol01-02` in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis vol01-02
```

This command removes the plex `vol01-02` and all associated subdisks.

Alternatively, you can first dissociate the plex and subdisks, and then remove them with the following commands:

```
# vxplex [-g diskgroup] dis plex
# vxedit [-g diskgroup] -r rm plex
```

When used together, these commands produce the same result as the `vxplex -o rm dis` command. The `-r` option to `vxedit rm` recursively removes all objects from the specified object downward. In this way, a plex and its associated subdisks can be removed by a single `vxedit` command.

Changing plex attributes

Caution: Change plex attributes with extreme care.

The `vxedit` command changes the attributes of plexes and other volume Manager objects. To change plex attributes, use the following command:

```
# vxedit [-g diskgroup] set attribute=value ... plex
```

Plex fields that can be changed using the `vxedit` command include:

- `name`
- `putil`*n*
- `tutil`*n*
- `comment`

The `putil`*n* field attributes are maintained on reboot; `tutil`*n* fields are temporary and are not retained on reboot. VxVM sets the `putil0` and `tutil0` utility fields. Other Symantec products, such as the Veritas Enterprise Administrator (VEA), set the `putil1` and `tutil1` fields. The `putil2` and `tutil2` are available for you to use for site-specific purposes.

The following example command sets the comment field, and also sets `tutil2` to indicate that the subdisk is in use:

```
# vxedit -g mydg set comment="plex comment" tutil2="u" vol01-02
```

To prevent a particular plex from being associated with a volume, set the `putil0` field to a non-null string, as shown in the following command:

```
# vxedit -g mydg set putil0="DO-NOT-USE" vol01-02
```

See the `vxedit`(1M) manual page for more information about using the `vxedit` command to change the attribute fields of VxVM objects.

Chapter 7

Creating volumes

This chapter describes how to create *volumes* in Veritas Volume Manager (VxVM). Volumes are logical devices that appear as physical disk partition devices to data management systems. Volumes enhance recovery from hardware failure, data availability, performance, and storage configuration.

Note: You can also use the Veritas Intelligent Storage Provisioning (ISP) feature to create and administer application volumes. These volumes are very similar to the traditional VxVM volumes that are described in this chapter. However, there are significant differences between the functionality of the two types of volume that prevents them from being used interchangeably. Refer to the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for more information about creating and administering ISP application volumes.

Volumes are created to take advantage of the VxVM concept of virtual disks. A file system can be placed on the volume to organize the disk space with files and directories. In addition, you can configure applications such as databases to organize data on volumes.

Note: Disks and disk groups must be initialized and defined to VxVM before volumes can be created from them. See "Administering disks" on page 77 and "Creating and administering disk groups" on page 165 for more information.

Types of volume layouts

VxVM allows you to create volumes with the following layout types:

Concatenated	A volume whose subdisks are arranged both sequentially and contiguously within a plex. Concatenation allows a volume to be created from multiple regions of one or more disks if there is not enough space for an entire volume on a single region of a disk. For more information, see “Concatenation and spanning” on page 35.
Striped	A volume with data spread evenly across multiple disks. *Stripes* are equal-sized fragments that are allocated alternately and evenly to the subdisks of a single plex. There must be at least two subdisks in a striped plex, each of which must exist on a different disk. Throughput increases with the number of disks across which a plex is striped. Striping helps to balance I/O load in cases where high traffic areas exist on certain subdisks. For more information, see “Striping (RAID-0)” on page 38.
Mirrored	A volume with multiple data plexes that duplicate the information contained in a volume. Although a volume can have a single data plex, at least two are required for true mirroring to provide redundancy of data. For the redundancy to be useful, each of these data plexes should contain disk space from different disks. For more information, see “Mirroring (RAID-1)” on page 42.
RAID-5	A volume that uses striping to spread data and parity evenly across multiple disks in an array. Each stripe contains a parity stripe unit and data stripe units. Parity can be used to reconstruct data if one of the disks fails. In comparison to the performance of striped volumes, write throughput of RAID-5 volumes decreases since parity information needs to be updated each time data is modified. However, in comparison to mirroring, the use of parity to implement data redundancy reduces the amount of space required. For more information, see “RAID-5 (striping with parity)” on page 45.
Mirrored-stripe	A volume that is configured as a striped plex and another plex that mirrors the striped one. This requires at least two disks for striping and one or more other disks for mirroring (depending on whether the plex is simple or striped). The advantages of this layout are increased performance by spreading data across multiple disks and redundancy of data. “Striping plus mirroring (mirrored-stripe or RAID-0+1)” on page 42.

Layered Volume

A volume constructed from other volumes. Non-layered volumes are constructed by mapping their subdisks to VM disks. Layered volumes are constructed by mapping their subdisks to underlying volumes (known as *storage volumes*), and allow the creation of more complex forms of logical layout. Examples of layered volumes are *striped-mirror* and *concatenated-mirror* volumes.

See "Layered volumes" on page 51.

A striped-mirror volume is created by configuring several mirrored volumes as the columns of a striped volume. This layout offers the same benefits as a non-layered mirrored-stripe volume. In addition it provides faster recovery as the failure of single disk does not force an entire striped plex offline.

See "Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)" on page 43.

A concatenated-mirror volume is created by concatenating several mirrored volumes. This provides faster recovery as the failure of a single disk does not force the entire mirror offline.

Supported volume logs and maps

Veritas Volume Manager supports the use of several types of logs and maps with volumes:

- FastResync Maps are used to perform quick and efficient resynchronization of mirrors (see "FastResync" on page 66 for details). These maps are supported either in memory (Non-Persistent FastResync), or on disk as part of a DCO volume (Persistent FastResync). Two types of DCO volume are supported:
 - Version 0 DCO volumes only support Persistent FastResync for the traditional third-mirror break-off type of volume snapshot. See "Version 0 DCO volume layout" on page 69, and "Creating a volume with a version 0 DCO volume" on page 250 for more information.
 - Version 20 DCO volumes, introduced in VxVM 4.0, support DRL logging (see below) and Persistent FastResync for full-sized and space-optimized instant volume snapshots. See "Version 20 DCO volume layout" on page 69, and "Creating a volume with a version 20 DCO volume" on page 252 for more information.

 See "Enabling FastResync on a volume" on page 292 for information on how to enable Persistent or Non-Persistent FastResync on a volume.
- Dirty region logs allow the fast recovery of mirrored volumes after a system crash (see "Dirty region logging" on page 60 for details). These logs are supported either as DRL log plexes, or as part of a version 20 DCO volume.

Refer to the following sections for information on creating a volume on which DRL is enabled:

- "Creating a volume with dirty region logging enabled" on page 252 for creating a volume with DRL log plexes.
- "Creating a volume with a version 20 DCO volume" on page 252 for creating a volume with DRL configured within a version 20 DCO volume.

- RAID-5 logs are used to prevent corruption of data during recovery of RAID-5 volumes (see "RAID-5 logging" on page 50 for details). These logs are configured as plexes on disks other than those that are used for the columns of the RAID-5 volume.
 See "Creating a RAID-5 volume" on page 256 for information on creating a RAID-5 volume together with RAID-5 logs.

Creating a volume

You can create volumes using an *advanced* approach, an *assisted* approach, or the rule-based storage allocation approach that is provided by the Intelligent Storage Provisioning (ISP) feature. Each method uses different tools. You may switch between the advanced and the assisted approaches at will. For more information about ISP, see the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide*.

Note: Most VxVM commands require superuser or equivalent privileges.

Advanced approach

The advanced approach consists of a number of commands that typically require you to specify detailed input. These commands use a "building block" approach that requires you to have a detailed knowledge of the underlying structure and components to manually perform the commands necessary to accomplish a certain task. Advanced operations are performed using several different VxVM commands.

To create a volume using the advanced approach

1. Create subdisks using `vxmake sd`; see "Creating subdisks" on page 215.
2. Create plexes using `vxmake plex`, and associate subdisks with them; see "Creating plexes" on page 223, "Associating subdisks with plexes" on page 218 and "Creating a volume using vxmake" on page 258.

3 Associate plexes with the volume using `vxmake vol`; see "Creating a volume using vxmake" on page 258.

4 Initialize the volume using `vxvol start` or `vxvol init zero`; see "Initializing and starting a volume created using vxmake" on page 261.

See "Creating a volume using a vxmake description file" on page 259 for an example of how you can combine steps 1 through 3 using a volume description file with `vxmake`.

See "Creating a volume using vxmake" on page 258 for an example of how to perform steps 2 and 3 to create a RAID-5 volume.

Assisted approach

The assisted approach takes information about what you want to accomplish and then performs the necessary underlying tasks. This approach requires only minimal input from you, but also permits more detailed specifications.

Assisted operations are performed primarily through the `vxassist` command or the Veritas Enterprise Administrator (VEA). `vxassist` and the VEA create the required plexes and subdisks using only the basic attributes of the desired volume as input. Additionally, they can modify existing volumes while automatically modifying any underlying or associated objects.

Both `vxassist` and the VEA use default values for many volume attributes, unless you provide specific values. They do not require you to have a thorough understanding of low-level VxVM concepts, `vxassist` and the VEA do not conflict with other VxVM commands or preclude their use. Objects created by `vxassist` and the VEA are compatible and inter-operable with objects created by other VxVM commands and interfaces.

For more information about the VEA, see the *Veritas Enterprise Administrator User's Guide* and VEA online help.

Using vxassist

You can use the `vxassist` utility to create and modify volumes. Specify the basic requirements for volume creation or modification, and `vxassist` performs the necessary tasks.

The advantages of using `vxassist` rather than the advanced approach include:

- Most actions require that you enter only one command rather than several.
- You are required to specify only minimal information to `vxassist`. If necessary, you can specify additional parameters to modify or control its actions.

- Operations result in a set of configuration changes that either succeed or fail as a group, rather than individually. System crashes or other interruptions do not leave intermediate states that you have to clean up. If `vxassist` finds an error or an exceptional condition, it exits after leaving the system in the same state as it was prior to the attempted `operation`.

The `vxassist` utility helps you perform the following tasks:

- Creating volumes.
- Creating mirrors for existing volumes.
- Growing or shrinking existing volumes.
- Backing up volumes online.
- Reconfiguring a volume's layout online.

`vxassist` obtains most of the information it needs from sources other than your input. `vxassist` obtains information about the existing objects and their layouts from the objects themselves.

For tasks requiring new disk space, `vxassist` seeks out available disk space and allocates it in the configuration that conforms to the layout specifications and that offers the best use of free space.

The `vxassist` command takes this form:

```
# vxassist [options] keyword volume [attributes...]
```

where *keyword* selects the task to perform. The first argument after a `vxassist` keyword, *volume*, is a volume name, which is followed by a set of desired volume attributes. For example, the keyword `make` allows you to create a new volume:

```
# vxassist [options] make volume length [attributes]
```

The *length* of the volume can be specified in sectors, kilobytes, megabytes, or gigabytes using a suffix character of `s`, `k`, `m`, or **`g`**. If no suffix is specified, the size is assumed to be in sectors. See the `vxintro`(1M) manual page for more information on specifying units.

Additional attributes can be specified as appropriate, depending on the characteristics that you wish the volume to have. Examples are stripe unit width, number of columns in a RAID-5 or stripe volume, number of mirrors, number of logs, and log type.

Note: By default, the `vxassist` command creates volumes in a default disk group according to the rules given in "Rules for determining the default disk group" on page 168. To use a different disk group, specify the `-g` *diskgroup* option to `vxassist`.

For details of available `vxassist` keywords and attributes, refer to the `vxassist`(1M) manual page.

The section, "Creating a volume on any disk" on page 243 describes the simplest way to create a volume with default attributes. Later sections describe how to create volumes with specific attributes. For example, "Creating a volume on specific disks" on page 244 describes how to control how `vxassist` uses the available storage space.

Setting default values for vxassist

The default values that the `vxassist` command uses may be specified in the file `/etc/default/vxassist`. The defaults listed in this file take effect if you do not override them on the command line, or in an alternate defaults file that you specify using the `-d` option. A default value specified on the command line always takes precedence. `vxassist` also has a set of built-in defaults that it uses if it cannot find a value defined elsewhere.

Note: You must create the `/etc/default` directory and the `vxassist` default file if these do not already exist on your system.

The format of entries in a defaults file is a list of attribute-value pairs separated by new lines. These attribute-value pairs are the same as those specified as options on the `vxassist` command line. Refer to the `vxassist`(1M) manual page for details.

To display the default attributes held in the file `/etc/default/vxassist`, use the following form of the `vxassist` command:

```
# vxassist help showattrs
```

The following is a sample `vxassist` defaults file:

```
#  By default:
#  create unmirrored, unstriped volumes
#  allow allocations to span drives
#  with RAID-5 create a log, with mirroring don't create a log
#  align allocations on cylinder boundaries
    layout=nomirror,nostripe,span,nocontig,raid5log,noregionlog,
    diskalign

#  use the fsgen usage type, except when creating RAID-5 volumes
    usetype=fsgen

# allow only root access to a volume
    mode=u=rw,g=,o=
    user=root
    group=root

# when mirroring, create two mirrors
    nmirror=2

# for regular striping, by default create between 2 and 8 stripe
# columns
```

```
    max_nstripe=8
    min_nstripe=2

#   for RAID-5, by default create between 3 and 8 stripe columns
    max_nraid5stripe=8
    min_nraid5stripe=3

#   by default, create 1 log copy for both mirroring and RAID-5
volumes
    nregionlog=1
    nraid5log=1

#   by default, limit mirroring log lengths to 32Kbytes
    max_regionloglen=32k

#   use 64K as the default stripe unit size for regular volumes
    stripe_stwid=64k

#   use 16K as the default stripe unit size for RAID-5 volumes
    raid5_stwid=16k
```

Discovering the maximum size of a volume

To find out how large a volume you can create within a disk group, use the following form of the `vxassist` command:

```
# vxassist [-g diskgroup] maxsize layout=layout [attributes]
```

For example, to discover the maximum size RAID-5 volume with 5 columns and 2 logs that you can create within the disk group, `dgrp`, enter the following command:

```
# vxassist -g dgrp maxsize layout=raid5 nlog=2
```

You can use storage attributes if you want to restrict the disks that `vxassist` uses when creating volumes. See “Creating a volume on specific disks” on page 244 for more information.

Note: The maximum size of a VxVM volume that you can create is 256TB.

Disk group alignment constraints on volumes

Certain constraints apply to the length of volumes and to the numeric values of size attributes that apply to volumes. If a volume is created in a disk group that is compatible with the Cross-platform Data Sharing (CDS) feature, the volume’s length and the values of volume attributes that define the sizes of objects such as logs or stripe units, must be an integer multiple of the alignment value of 8 blocks (8 kilobytes). If the disk group is not compatible with the CDS feature, the volume’s length and attribute size values must be multiples of 1 block (1kilobyte).

To discover the value in blocks of the alignment that is set on a disk group, use this command:

```
# vxprint -g diskgroup -G -F %align
```

By default, `vxassist` automatically rounds up the volume size and attribute size values to a multiple of the alignment value. (This is equivalent to specifying the attribute `dgalign_checking=round` as an additional argument to the `vxassist` command.)

If you specify the attribute `dgalign_checking=strict` to `vxassist`, the command fails with an error if you specify a volume length or attribute size value that is not a multiple of the alignment value for the disk group.

Creating a volume on any disk

By default, the `vxassist make` command creates a concatenated volume that uses one or more sections of disk space. On a fragmented disk, this allows you to put together a volume larger than any individual section of free disk space available.

Note: To change the default layout, edit the definition of the `layout` attribute defined in the `/etc/default/vxassist` file.

If there is not enough space on a single disk, `vxassist` creates a spanned volume. A spanned volume is a concatenated volume with sections of disk space spread across more than one disk. A spanned volume can be larger than any disk on a system, since it takes space from more than one disk.

To create a concatenated, default volume, use the following form of the `vxassist` command:

```
# vxassist [-b] [-g diskgroup] make volume length
```

Note: Specify the `-b` option if you want to make the volume immediately available for use. See “Initializing and starting a volume” on page 260 for details.

For example, to create the concatenated volume `voldefault` with a length of 10 gigabytes in the default disk group:

```
# vxassist -b make voldefault 10g
```

Creating a volume on specific disks

VxVM automatically selects the disks on which each volume resides, unless you specify otherwise. If you want a volume to be created on specific disks, you must designate those disks to VxVM. More than one disk can be specified.

To create a volume on a specific disk or disks, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  [layout=layout] diskname ...
```

For example, to create the volume `volspec` with length 5 gigabytes on disks `mydg03` and `mydg04`, use the following command:

```
# vxassist -b -g mydg make volspec 5g mydg03 mydg04
```

The `vxassist` command allows you to specify storage attributes. These give you control over the devices, including disks, controllers and targets, which `vxassist` uses to configure a volume. For example, you can specifically exclude disk mydg05:

```
# vxassist -b -g mydg make volspec 5g !mydg05
```

or exclude all disks that are on controller c2:

```
# vxassist -b -g mydg make volspec 5g !ctlr:c2
```

or include only disks on controller `c1` except for target `t5`:

```
# vxassist -b -g mydg make volspec 5g ctlr:c1 !target:c1t5
```

If you want a volume to be created using only disks from a specific disk group, use the `-g` option to `vxassist`, for example:

```
# vxassist -g bigone -b make volmega 20g bigone10 bigone11
```

or alternatively, use the `diskgroup` attribute:

```
# vxassist -b make volmega 20g diskgroup=bigone bigone10 \
  bigone11
```

Note: Any storage attributes that you specify for use must belong to the disk group. Otherwise, `vxassist` will not use them to create a volume.

You can also use storage attributes to control how `vxassist` uses available storage, for example, when calculating the maximum size of a volume, when growing a volume or when removing mirrors or logs from a volume. The following example excludes disks `dgrp07` and `dgrp08` when calculating the maximum size of RAID-5 volume that `vxassist` can create using the disks in the disk group `dg`:

```
# vxassist -b -g dgrp maxsize layout=raid5 nlog=2 !dgrp07 \
  !dgrp08
```

See the `vxassist`(1M) manual page for more information about using storage attributes. It is also possible to control how volumes are laid out on the specified storage as described in the next section "Specifying ordered allocation of storage to volumes."

Specifying ordered allocation of storage to volumes

Ordered allocation gives you complete control of space allocation. It requires that the number of disks that you specify to the `vxassist` command must match the number of disks that are required to create a volume. The order in which you specify the disks to `vxassist` is also significant.

If you specify the `-o ordered` option to `vxassist` when creating a volume, any storage that you also specify is allocated in the following order:

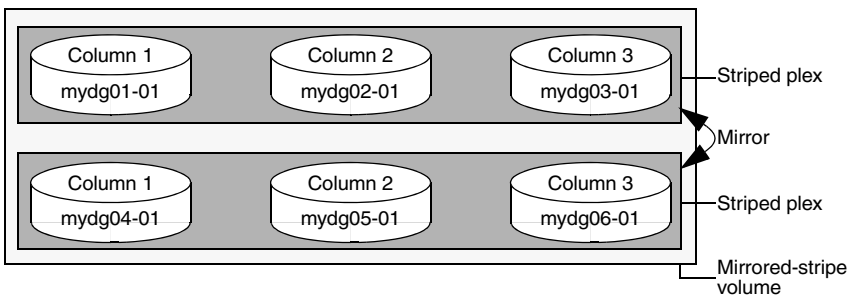
1. Concatenate disks.
2. Form columns.
3. Form mirrors.

For example, the following command creates a mirrored-stripe volume with 3 columns and 2 mirrors on 6 disks in the disk group, `mydg`:

```
# vxassist -b -g mydg -o ordered make mirstrvol 10g \
  layout=mirror-stripe ncol=3 \
  mydg01 mydg02 mydg03 mydg04 mydg05 mydg06
```

This command places columns 1, 2 and 3 of the first mirror on disks `mydg01`, `mydg02` and `mydg03` respectively, and columns 1, 2 and 3 of the second mirror on disks `mydg04`, `mydg05` and `mydg06` respectively. This arrangement is illustrated in Figure 7-1.

Figure 7-1 Example of using ordered allocation to create a mirrored-stripe volume

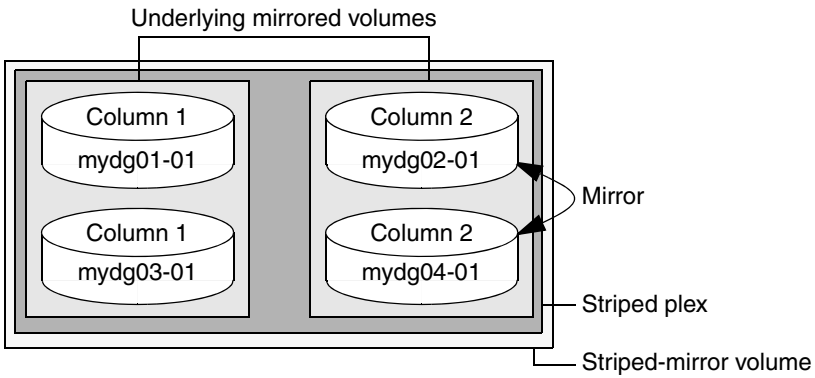


For layered volumes, `vxassist` applies the same rules to allocate storage as for non-layered volumes. For example, the following command creates a striped-mirror volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmirvol 10g \
  layout=stripe-mirror ncol=2 mydg01 mydg02 mydg03 mydg04
```

This command mirrors column 1 across disks `mydg01` and `mydg03`, and column 2 across disks `mydg02` and `mydg04`, as illustrated in Figure 7-2.

Figure 7-2 Example of using ordered allocation to create a striped-mirror volume

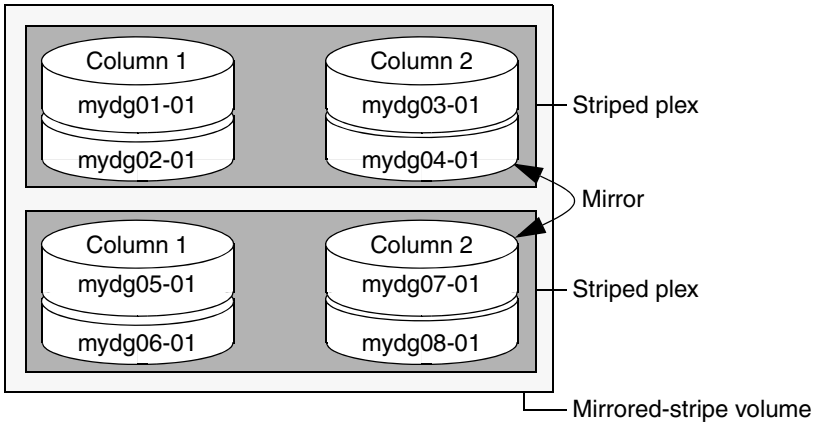


Additionally, you can use the `col_switch` attribute to specify how to concatenate space on the disks into columns. For example, the following command creates a mirrored-stripe volume with 2 columns:

```
# vxassist -b -g mydg -o ordered make strmir2vol 10g \
  layout=mirror-stripe ncol=2 col_switch=3g,2g \
  mydg01 mydg02 mydg03 mydg04 mydg05 mydg06 mydg07 mydg08
```

This command allocates 3 gigabytes from `mydg01` and 2 gigabytes from `mydg02` to column 1, and 3 gigabytes from `mydg03` and 2 gigabytes from `mydg04` to column 2. The mirrors of these columns are then similarly formed from disks `mydg05` through `mydg08`. This arrangement is illustrated in Figure 7-3.

Figure 7-3 Example of using concatenated disk space to create a mirrored-stripe volume

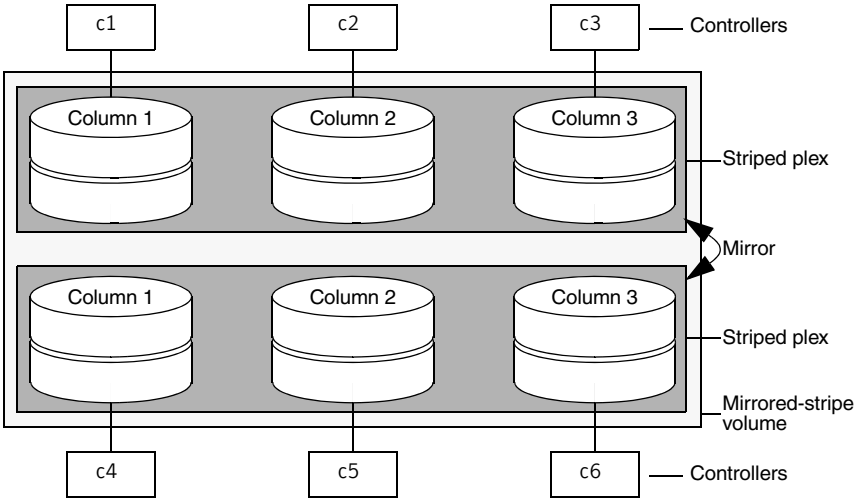


Other storage specification classes for controllers, enclosures, targets and trays can be used with ordered allocation. For example, the following command creates a 3-column mirrored-stripe volume between specified controllers:

```
# vxassist -b -g mydg -o ordered make mirstr2vol 80g \
  layout=mirror-stripe ncol=3 \
  ctlr:c1 ctlr:c2 ctlr:c3 ctlr:c4 ctlr:c5 ctlr:c6
```

This command allocates space for column 1 from disks on controllers `c1`, for column 2 from disks on controller `c2`, and so on as illustrated in Figure 7-4.

Figure 7-4 Example of storage allocation used to create a mirrored-stripe volume across controllers



For other ways in which you can control how `vxassist` lays out mirrored volumes across controllers, see “Mirroring across targets, controllers or enclosures” on page 255.

Creating a mirrored volume

Note: You need a full license to use this feature.

A mirrored volume provides data redundancy by containing more than one copy of its data. Each copy (or mirror) is stored on different disks from the original copy of the volume and from other mirrors. Mirroring a volume ensures that its data is not lost if a disk in one of its component mirrors fails.

Note: A mirrored volume requires space to be available on at least as many disks in the disk group as the number of mirrors in the volume.

To create a new mirrored volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror [nmirror=number] [init=active]
```

For example, to create the mirrored volume, `volmir`, in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volmir 5g layout=mirror
```

To create a volume with 3 instead of the default of 2 mirrors, modify the command to read:

```
# vxassist -b -g mydg make volmir 5g layout=mirror nmirror=3
```

Creating a mirrored-concatenated volume

Note: You need a full license to use this feature.

A mirrored-concatenated volume mirrors several concatenated plexes. To create a concatenated-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror-concat [nmirror=number]
```

Alternatively, first create a concatenated volume, and then mirror it as described in “Adding a mirror to a volume” on page 271.

Creating a concatenated-mirror volume

Note: You need a full license to use this feature.

A concatenated-mirror volume is an example of a layered volume which concatenates several underlying mirror volumes. To create a concatenated-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=concat-mirror [nmirror=number]
```

Creating a volume with a version 0 DCO volume

If a data change object (DCO) and DCO volume are associated with a volume, this allows Persistent FastResync to be used with the volume. (See "How persistent FastResync works with snapshots" on page 70 for details of how Persistent FastResync performs fast resynchronization of snapshot mirrors when they are returned to their original volume.)

Note: The procedure described in this section creates a volume with a data change object (DCO) and DCO volume that has a version 0 layout as introduced in VxVM 3.2. The version 0 layout supports traditional (third-mirror) snapshots, but not full-sized instant snapshots, space-optimized instant snapshots nor DRL configured within the DCO volume. See "Version 0 DCO volume layout" on page 69 and "Version 20 DCO volume layout" on page 69 for a description of the differences between the old and new DCO volume layouts.

For details of how to configure a volume with a version 20 DCO and DCO volume, see "Creating a volume with a version 20 DCO volume" on page 252. This is the preferred and recommended method.

See "Determining the DCO version number" on page 277 for details of how to determine the version number of a volume's DCO.

To perform fast resynchronization of mirrors after a system crash or reboot, you must also enable dirty region logging (DRL) on a mirrored volume. To add a DCO object and DCO volume to a volume on which DRL logging is enabled, follow the procedure described in "Adding a version 0 DCO and DCO volume" on page 356.

Note: You need a Veritas FlashSnap™ or FastResync license to use the Persistent FastResync feature. Even if you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. For more information about snap objects, see "How persistent FastResync works with snapshots" on page 70.

To create a volume with an attached version 0 DCO object and volume

1 Ensure that the disk group has been upgraded to version 90. Use the following command to check the version of a disk group:

```
# vxdg list diskgroup
```

To upgrade a disk group to version 90, use the following command:

```
# vxdg -T 90 upgrade diskgroup
```

For more information, see "Upgrading a disk group" on page 208.

2 Use the following command to create the volume (you may need to specify additional attributes to create a volume with the desired characteristics):

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=dco [ndcomirror=number] [dcolen=size] \
  [fastresync=on] [other attributes]
```

For non-layered volumes, the default number of plexes in the mirrored DCO volume is equal to the lesser of the number of plexes in the data volume or 2. For layered volumes, the default number of DCO plexes is always 2. If required, use the `ndcomirror` attribute to specify a different *number*. It is recommended that you configure as many DCO plexes as there are data plexes in the volume. For example, specify `ndcomirror=3` when creating a 3-way mirrored volume.

The default size of each plex is 132 blocks unless you use the `dcolen` attribute to specify a different *size*. If specified, the size of the plex must be a multiple of 33 blocks from 33 up to a maximum of 2112 blocks.

By default, FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute if you want to enable FastResync on the volume. If a DCO object and DCO volume are associated with the volume, Persistent FastResync is enabled; otherwise, Non-Persistent FastResync is enabled.

3 To enable DRL or sequential DRL logging on the newly created volume, use the following command:

```
# vxvol [-g diskgroup] set logtype=drl|drlseq volume
```

For more information, see the `vxassist`(1M) and `vxvol`(1M) manual pages.

If you use ordered allocation when creating a mirrored volume on specified storage, you can use the optional `logdisk` attribute to specify on which disks dedicated log plexes should be created. Use the following form of the `vxassist` command to specify the disks from which space for the logs is to be allocated:

```
# vxassist [-g diskgroup] -o ordered make volume length \
  layout=mirror logtype=log_type logdisk=disk[,disk,...] \
  storage_attributes
```

If you do not specify the `logdisk` attribute, `vxassist` locates the logs in the data plexes of the volume.

For more information about ordered allocation, see "Specifying ordered allocation of storage to volumes" on page 245 and the `vxassist`(1M) manual page.

Creating a volume with a version 20 DCO volume

To create a volume with an attached version 20 DCO object and volume

1. Ensure that the disk group has been upgraded to the latest version. Use the following command to check the version of a disk group:

   ```
   # vxdg list diskgroup
   ```

 To upgrade a disk group to the most recent version, use the following command:

   ```
   # vxdg upgrade diskgroup
   ```

 For more information, see "Upgrading a disk group" on page 208.

2. Use the following command to create the volume (you may need to specify additional attributes to create a volume with the desired characteristics):

   ```
   # vxassist [-g diskgroup] make volume length layout=layout \
     logtype=dco dcoversion=20 [drl=on|sequential|off] \
     [ndcomirror=number] [fastresync=on] [other attributes]
   ```

 Set the value of the `drl` attribute to `on` if dirty region logging (DRL) is to be used with the volume (this is the default setting). For a volume that will be written to sequentially, such as a database log volume, set the value to `sequential` to enable sequential DRL. The DRL logs are created in the DCO volume. The redundancy of the logs is determined by the number of mirrors that you specify using the `ndcomirror` attribute.

 By default, Persistent FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute if you want to enable Persistent FastResync on the volume.

For more information, see the `vxassist`(1M) manual page.

Note: See "Determining the DCO version number" on page 277 for details of how to determine the version number of a volume's DCO.

Creating a volume with dirty region logging enabled

Note: The procedure in this section is applicable to volumes that are created in disk groups with a version number of less than 110. To enable DRL or sequential DRL on a volume that is created within a disk group with a version number of 110 or greater, follow the procedure described in "Creating a volume with a version 20 DCO volume" on page 252, which creates the DRL logs within the plexes of a version 20 DCO volume.

Dirty region logging (DRL), if enabled, speeds recovery of mirrored volumes after a system crash. To enable DRL on a volume that is created within a disk group with a version number between 20 and 100, specify the `logtype=drl` attribute to the `vxassist make` command as shown in this example usage:

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drl [nlog=n] [loglen=size] [other attributes]
```

The `nlog` attribute can be used to specify the number of log plexes to add. By default, one log plex is added. The `loglen` attribute specifies the size of the log, where each bit represents one region in the volume. For example, the size of the log would need to be 20K for a 10GB volume with a region size of 64 kilobytes.

For example, to create a mirrored 10GB volume, `vol02`, with two log plexes in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg make vol02 10g layout=mirror logtype=drl \
  nlog=2 nmirror=2
```

Sequential DRL limits the number of dirty regions for volumes that are written to sequentially, such as database replay logs. To enable sequential DRL on a volume that is created within a disk group with a version number between 70 and 100, specify the `logtype=drlseq` attribute to the `vxassist make` command.

```
# vxassist [-g diskgroup] make volume length layout=layout \
  logtype=drlseq [nlog=n] [other attributes]
```

Note: If you also want to allow the use of Persistent FastResync with the volume, use the procedure described in "Creating a volume with a version 0 DCO volume" on page 250.

Creating a striped volume

A striped volume contains at least one plex that consists of two or more subdisks located on two or more physical disks. For more information on striping, see "Striping (RAID-0)" on page 38.

Note: A striped volume requires space to be available on at least as many disks in the disk group as the number of columns in the volume.

To create a striped volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=stripe
```

For example, to create the 10-gigabyte striped volume `volzebra`, in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volzebra 10g layout=stripe
```

This creates a striped volume with the default stripe unit size (64 kilobytes) and the default number of stripes (2).

You can specify the disks on which the volumes are to be created by including the disk names on the command line. For example, to create a 30-gigabyte striped volume on three specific disks, `mydg03`, `mydg04`, and `mydg05`, use the following command:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  mydg03 mydg04 mydg05
```

To change the number of columns or the stripe width, use the `ncolumn` and `stripeunit` modifiers with `vxassist`. For example, the following command creates a striped volume with 5 columns and a 32-kilobyte stripe size:

```
# vxassist -b -g mydg make stripevol 30g layout=stripe \
  stripeunit=32k ncol=5
```

Creating a mirrored-stripe volume

A mirrored-stripe volume mirrors several striped data plexes.

Note: A mirrored-stripe volume requires space to be available on at least as many disks in the disk group as the number of mirrors multiplied by the number of columns in the volume.

To create a striped-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=mirror-stripe [nmirror=number_mirrors] \
  [ncol=number_of_columns] [stripewidth=size]
```

Alternatively, first create a striped volume, and then mirror it as described in “Adding a mirror to a volume” on page 271. In this case, the additional data plexes may be either striped or concatenated.

Creating a striped-mirror volume

A striped-mirror volume is an example of a layered volume which stripes several underlying mirror volumes.

Note: A striped-mirror volume requires space to be available on at least as many disks in the disk group as the number of columns multiplied by the number of stripes in the volume.

To create a striped-mirror volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=stripe-mirror [nmirror=number_mirrors] \
  [ncol=number_of_columns] [stripewidth=size]
```

By default, VxVM attempts to create the underlying volumes by mirroring subdisks rather than columns if the size of each column is greater than the value

for the attribute `stripe-mirror-col-split-trigger-pt` that is defined in the `vxassist` defaults file.

If there are multiple subdisks per column, you can choose to mirror each subdisk individually instead of each column. To mirror at the subdisk level, specify the layout as `stripe-mirror-sd` rather than `stripe-mirror`. To mirror at the column level, specify the layout as `stripe-mirror-col` rather than `stripe-mirror`.

Mirroring across targets, controllers or enclosures

To create a volume whose mirrored data plexes lie on different controllers (also known as *disk duplexing*) or in different enclosures, use the `vxassist` command as described in this section.

In the following command, the attribute `mirror=target` specifies that volumes should be mirrored between identical target IDs on different controllers.

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=layout mirror=target [attributes]
```

The attribute `mirror=ctlr` specifies that disks in one mirror should not be on the same controller as disks in other mirrors within the same volume:

```
# vxassist [-b] [-g diskgroup] make volume length \
  layout=layout mirror=ctlr [attributes]
```

Note: Both paths of an active/passive array are not considered to be on different controllers when mirroring across controllers.

The following command creates a mirrored volume with two data plexes in the disk group, `mydg`:

```
# vxassist -b -g mydg make volspec 10g layout=mirror nmirror=2 \
  mirror=ctlr ctlr:c2 ctlr:c3
```

The disks in one data plex are all attached to controller `c2`, and the disks in the other data plex are all attached to controller `c3`. This arrangement ensures continued availability of the volume should either controller fail.

The attribute `mirror=enclr` specifies that disks in one mirror should not be in the same enclosure as disks in other mirrors within the same volume.

The following command creates a mirrored volume with two data plexes:

```
# vxassist -b make -g mydg volspec 10g layout=mirror nmirror=2 \
  mirror=enclr enclr:enc1 enclr:enc2
```

The disks in one data plex are all taken from enclosure `enc1`, and the disks in the other data plex are all taken from enclosure `enc2`. This arrangement ensures continued availability of the volume should either enclosure become unavailable.

See "Specifying ordered allocation of storage to volumes" on page 245 for a description of other ways in which you can control how volumes are laid out on the specified storage.

Creating a RAID-5 volume

Note: VxVM supports this feature for private disk groups, but not for shareable disk groups in a cluster environment.

A RAID-5 volume requires space to be available on at least as many disks in the disk group as the number of columns in the volume. Additional disks may be required for any RAID-5 logs that are created.

You need a full license to use this feature.

You can create RAID-5 volumes by using either the `vxassist` command (recommended) or the `vxmake` command. Both approaches are described below.

A RAID-5 volume contains a RAID-5 data plex that consists of three or more subdisks located on three or more physical disks. Only one RAID-5 data plex can exist per volume. A RAID-5 volume can also contain one or more RAID-5 log plexes, which are used to log information about data and parity being written to the volume. For more information on RAID-5 volumes, see "RAID-5 (striping with parity)" on page 45.

Caution: Do not create a RAID-5 volume with more than 8 columns because the volume will be unrecoverable in the event of the failure of more than one disk.

To create a RAID-5 volume, use the following command:

```
# vxassist [-b] [-g diskgroup] make volume length layout=raid5 \
  [ncol=number_of_columns] [stripewidth=size] [nlog=number] \
  [loglen=log_length]
```

For example, to create the RAID-5 volume `volraid` together with 2 RAID-5 logs in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg make volraid 10g layout=raid5 nlog=2
```

This creates a RAID-5 volume with the default stripe unit size on the default number of disks. It also creates two RAID-5 logs rather than the default of one log.

Note: If you require RAID-5 logs, you must use the `logdisk` attribute to specify the disks to be used for the log plexes.

RAID-5 logs can be concatenated or striped plexes, and each RAID-5 log associated with a RAID-5 volume has a complete copy of the logging information for the volume. To support concurrent access to the RAID-5 array, the log should be several times the stripe size of the RAID-5 plex.

It is suggested that you configure a minimum of two RAID-5 log plexes for each RAID-5 volume. These log plexes should be located on different disks. Having two RAID-5 log plexes for each RAID-5 volume protects against the loss of logging information due to the failure of a single disk.

If you use ordered allocation when creating a RAID-5 volume on specified storage, you must use the `logdisk` attribute to specify on which disks the RAID-5 log plexes should be created. Use the following form of the `vxassist` command to specify the disks from which space for the logs is to be allocated:

```
# vxassist [-b] [-g diskgroup] -o ordered make volume length \
  layout=raid5 [ncol=number_columns] [nlog=number] \
  [loglen=log_length] logdisk=disk[,disk,...] storage_attributes
```

For example, the following command creates a 3-column RAID-5 volume with the default stripe unit size on disks `mydg04`, `mydg05` and `mydg06`. It also creates two RAID-5 logs on disks `mydg07` and `mydg08`.

```
# vxassist -b -g mydg -o ordered make volraid 10g layout=raid5 \
  ncol=3 nlog=2 logdisk=mydg07,mydg08 mydg04 mydg05 mydg06
```

Note: The number of logs must equal the number of disks specified to `logdisk`.

For more information about ordered allocation, see "Specifying ordered allocation of storage to volumes" on page 245 and the `vxassist`(1M) manual page.

If you need to add more logs to a RAID-5 volume at a later date, follow the procedure described in "Adding a RAID-5 log" on page 283.

Creating tagged volumes

Volume tags are used to implement the Dynamic Storage Tiering feature of the Storage Foundation software. For more information about this feature, see the *Veritas File System Administrator's Guide*.

You can use the `tag` attribute with the `vxassist make` command to set a named tag and optional tag value on a volume, for example:

```
# vxassist -b -g mydg make volmir 5g layout=mirror tag=mirvol=5g
```

To list the tags that are associated with a volume, use this command:

```
# vxassist [-g diskgroup] listtag volume
```

To list the volumes that have a specified tag name, use this command:

```
# vxassist [-g diskgroup] list tag=tagname volume
```

Tag names and tag values are case-sensitive character strings of up to 256 characters. Tag names can consist of letters (A through Z and a through z), numbers (0 through 9), dashes (-), underscores (_) or periods (.) from the ASCII character set. A tag name must start with either a letter or an underscore. Tag values can consist of any character from the ASCII character set with a decimal value from 32 through 127. If a tag value includes any spaces, use the `vxassist settag` command to set the tag on the newly created volume.

Dotted tag hierarchies are understood by the `list` operation. For example, the listing for `tag=a.b` includes all volumes that have tag names that start with `a.b`.

The tag names `site`, `udid` and `vdid` are reserved and should not be used. To avoid possible clashes with future product features, it is recommended that tag names do not start with any of the following strings: `asl`, `be`, `isp`, `nbu`, `sf`, `symc` or `vx`.

See "Setting tags on volumes" on page 288.

Creating a volume using vxmake

As an alternative to using `vxassist`, you can create a volume using the `vxmake` command to arrange existing subdisks into plexes, and then to form these plexes into a volume. Subdisks can be created using the method described in "Creating subdisks" on page 215. The example given in this section is to create a RAID-5 volume using `vxmake`.

Creating a RAID-5 plex for a RAID-5 volume is similar to creating striped plexes, except that the `layout` attribute is set to `raid5`. Subdisks can be implicitly associated in the same way as with striped plexes. For example, to create a four-column RAID-5 plex with a stripe unit size of 32 sectors, use the following command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
  sd=mydg00-01,mydg01-00,mydg02-00,mydg03-00
```

Note that because four subdisks are specified, but the number of columns is not specified, the `vxmake` command assumes a four-column RAID-5 plex and places one subdisk in each column. Striped plexes are created using the same method except that the layout is specified as `stripe`. If the subdisks are to be created and added later, use the following command to create the plex:

```
# vxmake -g mydg plex raidplex layout=raid5 ncolumn=4 stwidth=32
```

Note: If no subdisks are specified, the `ncolumn` attribute must be specified. Subdisks can be added to the plex later using the `vxsd assoc` command (see "Associating subdisks with plexes" on page 218).

If each column in a RAID-5 plex is to be created from multiple subdisks which may span several physical disks, you can specify to which column each subdisk should be added. For example, to create a three-column RAID-5 plex using six subdisks, use the following form of the vxmake command:

```
# vxmake -g mydg plex raidplex layout=raid5 stwidth=32 \
  sd=mydg00-00:0,mydg01-00:1,mydg02-00:2,mydg03-00:0, \
  mydg04-00:1,mydg05-00:2
```

This command stacks subdisks mydg00-00 and mydg03-00 consecutively in column 0, subdisks mydg01-00 and mydg04-00 consecutively in column 1, and subdisks mydg02-00 and mydg05-00 in column 2. Offsets can also be specified to create sparse RAID-5 plexes, as for striped plexes.

Log plexes may be created as default concatenated plexes by not specifying a layout, for example:

```
# vxmake -g mydg plex raidlog1 sd=mydg06-00
# vxmake -g mydg plex raidlog2 sd=mydg07-00
```

The following command creates a RAID-5 volume, and associates the prepared RAID-5 plex and RAID-5 log plexes with it:

```
# vxmake -g mydg -Uraid5 vol raidvol \
  plex=raidplex,raidlog1,raidlog2
```

Note: Each RAID-5 volume has one RAID-5 plex where the data and parity are stored. Any other plexes associated with the volume are used as RAID-5 log plexes to log information about data and parity being written to the volume.

After creating a volume using vxmake, you must initialize it before it can be used. The procedure is described in "Initializing and starting a volume" on page 260.

Creating a volume using a vxmake description file

You can use the vxmake command to add a new volume, plex or subdisk to the set of objects managed by VxVM. vxmake adds a record for each new object to the VxVM configuration database. You can create records either by specifying parameters to vxmake on the command line, or by using a file which contains plain-text descriptions of the objects. The file can also contain commands for performing a list of tasks. Use the following form of the command to have vxmake read the file from the standard input:

```
# vxmake [-g diskgroup] < description_file
```

Alternatively, you can specify the file to vxmake using the -d option:

```
# vxmake [-g diskgroup] -d description_file
```

The following sample description file defines a volume, db, with two plexes, db-01 and db-02:

```
#rty  #name       #options
sd    mydg03-01   disk=mydg03 offset=0 len=10000
sd    mydg03-02   disk=mydg03 offset=25000 len=10480
sd    mydg04-01   disk=mydg04 offset=0 len=8000
sd    mydg04-02   disk=mydg04 offset=15000 len=8000
sd    mydg04-03   disk=mydg04 offset=30000 len=4480
plex  db-01       layout=STRIPE ncolumn=2 stwidth=16k
                  sd=mydg03-01:0/0,mydg03-02:0/10000,mydg04-01:1/0,
mydg04-02:1/8000,mydg04-03:1/16000
sd    ramd1-01    disk=ramd1 len=640
                  comment="Hot spot for dbvol
plex  db-02       sd=ramd1-01:40320
vol   db          usetype=gen plex=db-01,db-02
                  readpol=prefer prefname=db-02
                  comment="Uses mem1 for hot spot in last 5m
```

Note: The subdisk definition for plex, db-01, must be specified on a single line. It is shown here split across two lines because of space constraints.

The first plex, db-01, is striped and has five subdisks on two physical disks, mydg03 and mydg04. The second plex, db-02, is the preferred plex in the mirror, and has one subdisk, ramd1-01, on a volatile memory disk.

For detailed information about how to use vxmake, refer to the vxmake(1M) manual page.

After creating a volume using vxmake, you must initialize it before it can be used. The procedure is described in "Initializing and starting a volume created using vxmake" on page 261.

Initializing and starting a volume

If you create a volume using the vxassist command, vxassist initializes and starts the volume automatically unless you specify the attribute init=none.

When creating a volume, you can make it immediately available for use by specifying the -b option to the vxassist command, as shown here:

```
# vxassist -b [-g diskgroup] make volume length layout=mirror
```

The -b option makes VxVM carry out any required initialization as a background task. It also greatly speeds up the creation of striped volumes by initializing the columns in parallel.

As an alternative to the -b option, you can specify the init=active attribute to make a new volume immediately available for use. In this example, init=active is specified to prevent VxVM from synchronizing the empty data plexes of a new mirrored volume:

```
# vxassist [-g diskgroup] make volume length layout=mirror \
  init=active
```

Caution: There is a very small risk of errors occurring when the init=active attribute is used. Although written blocks are guaranteed to be consistent, read errors can arise in the unlikely event that fsck attempts to verify uninitialized space in the file system, or if a file remains uninitialized following a system crash. If in doubt, use the -b option to vxassist instead.

This command writes zeroes to the entire length of the volume and to any log plexes. It then makes the volume active. You can also zero out a volume by specifying the attribute init=zero to vxassist, as shown in this example:

```
# vxassist [-g diskgroup] make volume length layout=raid5 \
  init=zero
```

Note: You cannot use the -b option to make this operation a background task.

Initializing and starting a volume created using vxmake

A volume may be initialized by running the vxvol command if the volume was created by the vxmake command and has not yet been initialized, or if the volume has been set to an uninitialized state.

To initialize and start a volume, use the following command:

```
# vxvol [-g diskgroup] start volume
```

The following command can be used to enable a volume without initializing it:

```
# vxvol [-g diskgroup] init enable volume
```

This allows you to restore data on the volume from a backup before using the following command to make the volume fully active:

```
# vxvol [-g diskgroup] init active volume
```

If you want to zero out the contents of an entire volume, use this command to initialize it:

```
# vxvol [-g diskgroup] init zero volume
```

Accessing a volume

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/`*diskgroup*`/`*volume*	block device file for *volume*
`/dev/vx/rdsk/`*diskgroup*`/`*volume*	character device file for *volume*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

Note: As the `rootdg` disk group no longer has special significance, VxVM only creates volume device nodes for this disk group in the `/dev/vx/dsk/rootdg` and `/dev/vx/rdsk/rootdg` directories. VxVM does not create device nodes in the `/dev/vx/dsk` or `/dev/vx/rdsk` directories for the `rootdg` disk group.

Chapter 8

Administering volumes

This chapter describes how to perform common maintenance tasks on volumes in Veritas Volume Manager (VxVM). This includes displaying volume information, monitoring tasks, adding and removing logs, resizing volumes, removing mirrors, removing volumes, and changing the layout of volumes without taking them offline.

Note: You can also use the Veritas Intelligent Storage Provisioning (ISP) feature to create and administer application volumes. These volumes are very similar to the traditional VxVM volumes that are described in this chapter. However, there are significant differences between the functionality of the two types of volumes that prevents them from being used interchangeably. Refer to the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for more information about creating and administering ISP application volumes.

Most VxVM commands require superuser or equivalent privileges.

Displaying volume information

You can use the `vxprint` command to display information about how a volume is configured.

To display the volume, plex, and subdisk record information for all volumes in the system, use the following command:

```
# vxprint -hvt
```

The `vxprint` command can also be applied to a single disk group:

```
# vxprint -g mydg -hvt
```

This is example output from this command:

```
V   NAME      RVG/VSET/CO KSTATE   STATE    LENGTH   READPOL    PREFPLEX   UTYPE
PL  NAME      VOLUME      KSTATE   STATE    LENGTH   LAYOUT     NCOL/WID   MODE
SD  NAME      PLEX        DISK     DISKOFFS LENGTH   [COL/]OFF  DEVICE     MODE
SV  NAME      PLEX        VOLNAME  NVOLLAYR LENGTH   [COL/]OFF  AM/NM      MODE
SC  NAME      PLEX        CACHE    DISKOFFS LENGTH   [COL/]OFF  DEVICE     MODE
DC  NAME      PARENTVOL   LOGVOL
SP  NAME      SNAPVOL     DCO

v   pubs      -           ENABLED  ACTIVE   22880    SELECT     -          fsgen
pl  pubs-01   pubs        ENABLED  ACTIVE   22880    CONCAT     -          RW
sd  mydg11-01 pubs-01     mydg11   0        22880    0            c1t0d0     ENA

v   voldef    -           ENABLED  ACTIVE   20480    SELECT     -          fsgen
pl  voldef-01 voldef      ENABLED  ACTIVE   20480    CONCAT     -          RW
sd  mydg12-02 voldef-0    mydg12   0        20480    0            c1t1d0     ENA
```

Here `v` is a volume, `pl` is a plex, and `sd` is a subdisk. The top few lines indicate the headers that match each type of output line that follows. Each volume is listed along with its associated plexes and subdisks.

Note: The headings for sub-volumes (`SV`), storage caches (`SC`), data change objects (`DCO`) and snappoints (`SP`) can be ignored here. No such objects are associated with these volumes.

To display volume-related information for a specific volume, use the following command:

```
# vxprint [-g diskgroup] -t volume
```

For example, to display information about the volume, `voldef`, in the disk group, `mydg`, use the following command:

```
# vxprint -g mydg -t voldef
```

This is example output from this command:

```
V  NAME         RVG/VSET/CO  KSTATE   STATE    LENGTH   READPOL    PREFPLEX  UTYPE

v  voldef       -            ENABLED  ACTIVE   20480    SELECT     -         fsgen
```

Note: If you enable enclosure-based naming, and use the `vxprint` command to display the structure of a volume, it shows enclosure-based disk device names (disk access names) rather than c#t#d# names. See "Discovering the association between enclosure and OS based disk names" on page 94 for information on how to obtain the true device names.

The following section describes the meaning of the various volume states that may be displayed.

Volume states

The following volume states may be displayed by VxVM commands such as `vxprint`:

ACTIVE volume state

The volume has been started (kernel state is currently ENABLED) or was in use (kernel state was ENABLED) when the machine was rebooted. If the volume is currently ENABLED, the state of its plexes at any moment is not certain (since the volume is in use).

If the volume is currently DISABLED, this means that the plexes cannot be guaranteed to be consistent, but are made consistent when the volume is started.

For a RAID-5 volume, if the volume is currently DISABLED, parity cannot be guaranteed to be synchronized.

CLEAN volume state

The volume is not started (kernel state is DISABLED) and its plexes are synchronized. For a RAID-5 volume, its plex stripes are consistent and its parity is good.

EMPTY volume state

The volume contents are not initialized. The kernel state is always DISABLED when the volume is EMPTY.

INVALID volume state

The contents of an instant snapshot volume no longer represent a true point-in-time image of the original volume.

NEEDSYNC volume state

The volume requires a resynchronization operation the next time it is started. For a RAID-5 volume, a parity resynchronization operation is required.

REPLAY volume state

The volume is in a transient state as part of a log replay. A log replay occurs when it becomes necessary to use logged parity and data. This state is only applied to RAID-5 volumes.

SYNC volume state

The volume is either in read-writeback recovery mode (kernel state is currently ENABLED) or was in read-writeback mode when the machine was rebooted (kernel state is DISABLED). With read-writeback recovery, plex consistency is recovered by reading data from blocks of one plex and writing the data to all other writable plexes. If the volume is ENABLED, this means that the plexes are being resynchronized through the read-writeback recovery. If the volume is DISABLED, it means that the plexes were being resynchronized through read-writeback when the machine rebooted and therefore still need to be synchronized.

For a RAID-5 volume, the volume is either undergoing a parity resynchronization (kernel state is currently ENABLED) or was having its parity resynchronized when the machine was rebooted (kernel state is DISABLED).

Note: The interpretation of these flags during volume startup is modified by the persistent state log for the volume (for example, the DIRTY/CLEAN flag). If the clean flag is set, an ACTIVE volume was not written to by any processes or was not even open at the time of the reboot; therefore, it can be considered CLEAN. The clean flag is always set in any case where the volume is marked CLEAN.

Volume kernel states

The *volume kernel state* indicates the accessibility of the volume. The volume kernel state allows a volume to have an offline (DISABLED), maintenance (DETACHED), or online (ENABLED) mode of operation.

Note: No user intervention is required to set these states; they are maintained internally. On a system that is operating properly, all volumes are ENABLED.

The following volume kernel states are defined:

DETACHED volume kernel state

Maintenance is being performed on the volume. The volume cannot be read from or written to, but certain plex operations and `ioctl` function calls are accepted.

DISABLED volume kernel state

The volume is offline and cannot be accessed.

ENABLED volume kernel state

The volume is online and can be read from or written to.

Monitoring and controlling tasks

Note: VxVM supports this feature for private disk groups, but not for shareable disk groups in a cluster environment.

The VxVM task monitor tracks the progress of system recovery by monitoring task creation, maintenance, and completion. The task monitor allows you to monitor task progress and to modify characteristics of tasks, such as pausing and recovery rate (for example, to reduce the impact on system performance).

Specifying task tags

Every task is given a unique *task identifier*. This is a numeric identifier for the task that can be specified to the `vxtask` utility to specifically identify a single task. Several VxVM utilities also provide a `-t` option to specify an alphanumeric tag of up to 16 characters in length. This allows you to group several tasks by associating them with the same tag.

The `vxassist`, `vxevac`, `vxplex`, `vxmirror`, `vxrecover`, `vxrelayout`, `vxresize`, `vxsd`, and `vxvol` utilities allow you to specify a tag using the `-t` option. For example, to execute a `vxrecover` command and track all the resulting tasks as a group with the task tag `myrecovery`, use the following command:

```
# vxrecover -g mydg -t myrecovery -b mydg05
```

Any tasks started by the utilities invoked by `vxrecover` also inherit its task ID and task tag, so establishing a parent-child task relationship.

For more information about the utilities that support task tagging, see their respective manual pages.

Managing tasks with vxtask

Note: New tasks take time to be set up, and so may not be immediately available for use after a command is invoked. Any script that operates on tasks may need to poll for the existence of a new task.

You can use the `vxtask` command to administer operations on VxVM tasks that are running on the system. Operations include listing tasks, modifying the state of a task (pausing, resuming, aborting) and modifying the rate of progress of a task. For detailed information about how to use `vxtask`, refer to the `vxtask`(1M) manual page.

VxVM tasks represent long-term operations in progress on the system. Every task gives information on the time the operation started, the size and progress of the operation, and the state and rate of progress of the operation. The administrator can change the state of a task, giving coarse-grained control over the progress of the operation. For those operations that support it, the rate of progress of the task can be changed, giving more fine-grained control over the task.

vxtask operations

The `vxtask` command supports the following operations:

`abort` Causes the specified task to cease operation. In most cases, the operations “back out” as if an I/O error occurred, reversing what has been done so far to the largest extent possible.

`list` Lists tasks running on the system in one-line summaries. The `-l` option prints tasks in long format. The `-h` option prints tasks hierarchically, with child tasks following the parent tasks. By default, all tasks running on the system are printed. If a `taskid` argument is supplied, the output is limited to those tasks whose `taskid` or task tag match `taskid`. The remaining arguments are used to filter tasks and limit the tasks actually listed.

`monitor` Prints information continuously about a task or group of tasks as task information changes. This allows you to track the progression of tasks. Specifying `-l` causes a long listing to be printed. By default, short one-line listings are printed. In addition to printing task information when a task state changes, output is also

generated when the task completes. When this occurs, the state of the task is printed as `EXITED`.

`pause` Puts a running task in the paused state, causing it to suspend operation.

`resume` Causes a paused task to continue operation.

`set` Changes modifiable parameters of a task. Currently, there is only one modifiable parameter, `slow[=`*iodelay*`]`, which can be used to reduce the impact that copy operations have on system performance. If `slow` is specified, this introduces a delay between such operations with a default value for *iodelay* of 250 milliseconds. The larger the value of *iodelay* that is specified, the slower is the progress of the task and the fewer system resources that it consumes in a given time. (The `slow` attribute is also accepted by the `vxplex`, `vxvol` and `vxrecover` commands.)

Using the vxtask command

To list all tasks currently running on the system, use the following command:

```
# vxtask list
```

To print tasks hierarchically, with child tasks following the parent tasks, specify the `-h` option, as follows:

```
# vxtask -h list
```

To trace all tasks in the disk group, `foodg`, that are currently paused, as well as any tasks with the tag `sysstart`, use the following command:

```
# vxtask -g foodg -p -i sysstart list
```

Use the `vxtask -p list` command lists all paused tasks, and use `vxtask resume` to continue execution (the task may be specified by its ID or by its tag):

```
# vxtask -p list
# vxtask resume 167
```

To monitor all tasks with the tag `myoperation`, use the following command:

```
# vxtask monitor myoperation
```

To cause all tasks tagged with `recovall` to exit, use the following command:

```
# vxtask abort recovall
```

This command causes VxVM to attempt to reverse the progress of the operation so far. For an example of how to use `vxtask` to monitor and modify the progress of the Online Relayout feature, see “Controlling the progress of a relayout” on page 299.

Stopping a volume

Stopping a volume renders it unavailable to the user, and changes the volume kernel state from ENABLED or DETACHED to DISABLED. If the volume cannot be disabled, it remains in its current state. To stop a volume, use the following command:

```
# vxvol [-g diskgroup] [-f] stop volume ...
```

To stop all volumes in a specified disk group, use the following command:

```
# vxvol [-g diskgroup] [-f] stopall
```

Caution: If you use the `-f` option to forcibly disable a volume that is currently open to an application, the volume remains open, but its contents are inaccessible. I/O operations on the volume fail, and this may cause data loss. It is not possible to deport a disk group until all of its volumes are closed.

If you need to prevent a closed volume from being opened, it is recommended that you use the `vxvol maint` command, as described in the following section.

Putting a volume in maintenance mode

If all mirrors of a volume become `STALE`, you can place the volume in maintenance mode. Then you can view the plexes while the volume is `DETACHED` and determine which plex to use for reviving the others. To place a volume in maintenance mode, use the following command:

```
# vxvol [-g diskgroup] maint volume
```

To assist in choosing the revival source plex, use `vxprint` to list the stopped volume and its plexes.

To take a plex (in this example, `vol01-02` in the disk group, `mydg`) offline, use the following command:

```
# vxmend -g mydg off vol01-02
```

The `vxmend on` command can change the state of an `OFFLINE` plex of a `DISABLED` volume to `STALE`. For example, to put a plex named `vol01-02` in the STALE state, use the following command:

```
# vxmend -g mydg on vol01-02
```

Running the `vxvol start` command on the volume then revives the plex as described in the next section.

Starting a volume

Starting a volume makes it available for use, and changes the volume state from DISABLED or DETACHED to ENABLED. To start a DISABLED or DETACHED volume, use the following command:

```
# vxvol [-g diskgroup] start volume ...
```

If a volume cannot be enabled, it remains in its current state.

To start all DISABLED or DETACHED volumes in a disk group, enter:

```
# vxvol -g diskgroup startall
```

Alternatively, to start a DISABLED volume, use the following command:

```
# vxrecover -g diskgroup -s volume ...
```

To start all DISABLED volumes, enter:

```
# vxrecover -s
```

To prevent any recovery operations from being performed on the volumes, additionally specify the `-n` option to `vxrecover`.

Adding a mirror to a volume

A mirror can be added to an existing volume with the `vxassist` command, as follows:

```
# vxassist [-b] [-g diskgroup] mirror volume
```

Note: If specified, the `-b` option makes synchronizing the new mirror a background task.

For example, to create a mirror of the volume `voltest` in the disk group, `mydg`, use the following command:

```
# vxassist -b -g mydg mirror voltest
```

Another way to mirror an existing volume is by first creating a plex, and then attaching it to a volume, using the following commands:

```
# vxmake [-g diskgroup] plex plex sd=subdisk ...
# vxplex [-g diskgroup] att volume plex
```

Mirroring all volumes

To mirror all volumes in a disk group to available disk space, use the following command:

```
# /etc/vx/bin/vxmirror -g diskgroup -a
```

To configure VxVM to create mirrored volumes by default, use the following command:

```
# /etc/vx/bin/vxmirror -d yes
```

If you make this change, you can still make unmirrored volumes by specifying `nmirror=1` as an attribute to the `vxassist` command. For example, to create an unmirrored 20-gigabyte volume named `nomirror` in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg make nomirror 20g nmirror=1
```

Mirroring volumes on a VM disk

Mirroring volumes on a VM disk gives you one or more copies of your volumes in another disk location. By creating mirror copies of your volumes, you protect your system against loss of data in case of a disk failure.

Note: This task only mirrors concatenated volumes. Volumes that are already mirrored or that contain subdisks that reside on multiple disks are ignored.

To mirror volumes on a disk

1. Make sure that the target disk has an equal or greater amount of space as the originating disk.
2. Select menu item `5 (Mirror volumes on a disk)` from the `vxdiskadm` main menu.
3. At the following prompt, enter the disk name of the disk that you wish to mirror:

```
Mirror volumes on a disk
Menu: VolumeManager/Disk/Mirror

This operation can be used to mirror volumes on a disk. These
volumes can be mirrored onto another disk or onto any
available disk space. Volumes will not be mirrored if they are
already mirrored. Also, volumes that are comprised of more
than one subdisk will not be mirrored.

Enter disk name [<disk>,list,q,?] mydg02
```

4. At the following prompt, enter the target disk name (this disk must be the same size or larger than the originating disk):

```
You can choose to mirror volumes from disk mydg02 onto any
available disk space, or you can choose to mirror onto a
specific disk. To mirror to a specific disk, select the name of
that disk. To mirror to any available disk space, select
"any".
Enter destination disk [<disk>,list,q,?] (default: any) mydg01
```

5 At the following prompt, press Return to make the mirror:

```
The requested operation is to mirror all volumes on disk
mydg02 in disk group mydg onto available disk space on disk
mydg01.

VxVM NOTICE V-5-2-229 This operation can take a long time to
complete.

Continue with operation? [y,n,q,?] (default: y)
```

The `vxdiskadm` program displays the status of the mirroring operation, as follows:

```
VxVM vxmirror INFO V-5-2-22 Mirror volume voltest-bk00 ...

VxVM INFO V-5-2-674 Mirroring of disk mydg01 is complete.
```

6 At the following prompt, indicate whether you want to mirror volumes on another disk (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Mirror volumes on another disk? [y,n,q,?] (default: n)
```

Removing a mirror

When a mirror is no longer needed, you can remove it to free up disk space.

Note: The last valid plex associated with a volume cannot be removed.

To remove a mirror from a volume, use the following command:

```
# vxassist [-g diskgroup] remove mirror volume
```

Additionally, you can use storage attributes to specify the storage to be removed. For example, to remove a mirror on disk `mydg01` from volume vol01, enter:

```
# vxassist -g mydg remove mirror vol01 !mydg01
```

For more information about storage attributes, see “Creating a volume on specific disks” on page 244.

Alternatively, use the following command to dissociate and remove a mirror from a volume:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove a mirror named `vol01-02` from the disk group, `mydg`, use the following command:

```
# vxplex -g mydg -o rm dis vol01-02
```

This command removes the mirror `vol01-02` and all associated subdisks. This is equivalent to entering the following separate commands:

```
# vxplex -g mydg dis vol01-02
# vxedit -g mydg -r rm vol01-02
```

Adding logs and maps to volumes

In Veritas Volume Manager, several types of volume logs and maps are supported:

- FastResync Maps are used to perform quick and efficient resynchronization of mirrors (see "FastResync" on page 66 for details). These maps are supported either in memory (Non-Persistent FastResync), or on disk as part of a DCO volume (Persistent FastResync). Two types of DCO volumes are supported:
 - Version 0 DCO volumes only support Persistent FastResync for the traditional third-mirror break-off type of volume snapshot. See "Version 0 DCO volume layout" on page 69, and "Adding a version 0 DCO and DCO volume" on page 356 for more information.
 - Version 20 DCO volumes, introduced in VxVM 4.0, support DRL logging (see below) and Persistent FastResync for full-sized and space-optimized instant volume snapshots. See "Version 20 DCO volume layout" on page 69, and "Preparing a volume for DRL and instant snapshots" on page 275 for more information.

 See "Enabling FastResync on a volume" on page 292 for information on how to enable Persistent or Non-Persistent FastResync on a volume.
- Dirty Region Logs allow the fast recovery of mirrored volumes after a system crash (see "Dirty region logging" on page 60 for details). These logs are supported either as DRL log plexes, or as part of a version 20 DCO volume. Refer to the following sections for information on enabling DRL on a volume:
 - "Adding traditional DRL logging to a mirrored volume" on page 281 describes how to add DRL log plexes to a volume.
 - "Preparing a volume for DRL and instant snapshots" on page 275 describes how to configure DRL for a volume using a version 20 DCO volume.
- RAID-5 logs are used to prevent corruption of data during recovery of RAID-5 volumes (see "RAID-5 logging" on page 50 for details). These logs are configured as plexes on disks other than those that are used for the columns of the RAID-5 volume.
 See "Adding a RAID-5 log" on page 283 for information on adding RAID-5 logs to a RAID-5 volume.

Preparing a volume for DRL and instant snapshots

Note: This procedure describes how to add a version 20 data change object (DCO) and DCO volume to a volume that you previously created in a disk group with a version number of 110 or greater. If you are creating a new volume in a disk group with a version number of 110 or greater, you can specify the co-creation of a DCO and DCO volume and enable DRL as described in "Creating a volume with a version 20 DCO volume" on page 252. If the volume was created in a release prior to VxVM 4.0, use the procedure in "Upgrading existing volumes to use version 20 DCOs" on page 279.

You need a full VxVM license and a Veritas FlashSnap™ or FastResync license to use the DRL and FastResync features. Even if you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. For more information about snap objects, see "How persistent FastResync works with snapshots" on page 70. See "Determining the DCO version number" on page 277 for details of how to determine the version number of a volume's DCO.

Use the following command to add a version 20 DCO and DCO volume to a volume:

```
# vxsnap [-g diskgroup] prepare volume [ndcomirs=number] \
  [regionsize=size] [drl=on|sequential|off] \
  [storage_attribute ...]
```

The `ndcomirs` attribute specifies the number of DCO plexes that are created in the DCO volume. It is recommended that you configure as many DCO plexes as there are data and snapshot plexes in the volume. The DCO plexes are used to set up a DCO volume for any snapshot volume that you subsequently create from the snapshot plexes. For example, specify `ndcomirs=5` for a volume with 3 data plexes and 2 snapshot plexes.

The value of the `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=on` (this is the default setting). If sequential DRL is required, specify `drl=sequential`. If DRL is not required, specify `drl=off`.

You can also specify storage attributes to define the disks that can and/or cannot be used for the plexes of the DCO volume. See "Specifying storage for version 20 DCO plexes" on page 276 for details.

Note: The `vxsnap prepare` command automatically enables Persistent FastResync on the volume. Persistent FastResync is also set automatically on any snapshots that are generated from a volume on which this feature is enabled.

If the volume is a RAID-5 volume, it is converted to a layered volume that can be used with instant snapshots and Persistent FastResync. See "Using a DCO and DCO volume with a RAID-5 volume" on page 277 for details.

By default, a version 20 DCO volume contains 32 per-volume maps. If you require more maps than this, you can use the `vxsnap addmap` command to add more maps. See the `vxsnap`(1M) manual page for details of this command.

Specifying storage for version 20 DCO plexes

If the disks that contain volumes and their snapshots are to be moved into different disk groups, you must ensure that the disks that contain their DCO plexes can accompany them. You can use storage attributes to specify which disks to use for the DCO plexes. (If you do not want to use dirty region logging (DRL) with a volume, you can specify the same disks as those on which the volume is configured, assuming that space is available on the disks). For example, to add a DCO object and mirrored DCO volume with plexes on `disk05` and `disk06` to the volume, `myvol`, use the following command:

```
# vxsnap -g mydg prepare myvol ndcomirs=2 alloc=disk05,disk06
```

To view the details of the DCO object and DCO volume that are associated with a volume, use the `vxprint` command. The following is example `vxprint -vh` output for the volume named `vol1` (the TUTIL0 and PUTIL0 columns are omitted for clarity):

```
TY   NAME         ASSOC       KSTATE   LENGTH  PLOFFS  STATE    ...
v    vol1         fsgen       ENABLED  1024    -       ACTIVE
pl   vol1-01      vol1        ENABLED  1024    -       ACTIVE
sd   disk01-01    vol1-01     ENABLED  1024    0       -
pl   foo-02       vol1        ENABLED  1024    -       ACTIVE
sd   disk02-01    vol1-02     ENABLED  1024    0       -
dc   vol1_dco     vol1        -        -       -       -
v    vol1_dcl     gen         ENABLED  132     -       ACTIVE
pl   vol1_dcl-01  vol1_dcl    ENABLED  132     -       ACTIVE
sd   disk03-01    vol1_dcl-01 ENABLED  132     0       -
pl   vol1_dcl-02  vol1_dcl    ENABLED  132     -       ACTIVE
sd   disk04-01    vol1_dcl-02 ENABLED  132     0       -
```

In this output, the DCO object is shown as `vol1_dco`, and the DCO volume as `vol1_dcl` with 2 plexes, `vol1_dcl-01` and `vol1_dcl-02`.

If required, you can use the `vxassist move` command to relocate DCO plexes to different disks. For example, the following command moves the plexes of the DCO volume, `vol1_dcl`, for volume `vol1` from `disk03` and `disk04` to `disk07` and `disk08`:

```
# vxassist -g mydg move vol1_dcl !disk03 !disk04 disk07 disk08
```

For more information, see "Moving DCO volumes between disk groups" on page 200, and the `vxassist`(1M) and `vxsnap`(1M) manual pages.

Using a DCO and DCO volume with a RAID-5 volume

The procedure in the previous section can be used to add a DCO and DCO volume to a RAID-5 volume. This allows you to use Persistent FastResync on the volume for fast resynchronization of snapshots on returning them to their original volume. However, the procedure has the side effect of converting the RAID-5 volume into a special type of layered volume. You can create space-optimized instant snapshots of such a volume, and you can add mirrors that may be broken off as full-sized instant snapshots. You cannot relayout or resize such a volume unless you convert it back to a pure RAID-5 volume.

To convert a volume back to a RAID-5 volume, remove any snapshot plexes from the volume, and dissociate the DCO and DCO volume from the layered volume using the procedure described in "Removing support for DRL and instant snapshots from a volume" on page 279. You can then perform relayout and resize operations on the resulting non-layered RAID-5 volume.

To allow Persistent FastResync to be used with the RAID-5 volume again, re-associate the DCO and DCO volume as described in "Preparing a volume for DRL and instant snapshots" on page 275.

Note: Dissociating a DCO and DCO volume disables FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

Determining the DCO version number

The instant snapshot and DRL-enabled DCO features require that a version 20 DCO be associated with a volume, rather than an earlier version 0 DCO.

To find out the version number of a DCO that is associated with a volume

1. Use the `vxprint` command on the volume to discover the name of its DCO:

   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
   ```

2. Use the `vxprint` command on the DCO to determine its version number:

   ```
   # vxprint [-g diskgroup] -F%version $DCONAME
   ```

Determining if DRL is enabled on a volume

To determine if DRL (configured using a version 20 DCO volume) is enabled on a volume

1. Use the `vxprint` command on the volume to discover the name of its DCO:
   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
   ```
2. To determine if DRL is enabled on the volume, use the following command with the volume's DCO:
   ```
   # vxprint [-g diskgroup] -F%drl $DCONAME
   ```
 DRL is enabled if this command displays `on`.
3. If DRL is enabled, use this command with the DCO to determine if sequential DRL is enabled:
   ```
   # vxprint [-g diskgroup] -F%sequentialdrl $DCONAME
   ```
 Sequential DRL is enabled if this command displays `on`.

Alternatively, you can use this command with the volume:

```
# vxprint [-g diskgroup] -F%log_type volume
```

This displays the logging type as `REGION` for DRL, `DRLSEQ` for sequential DRL, or `NONE` if DRL is not enabled.

Note: If the number of active mirrors in the volume is less than 2, DRL logging is not performed even if DRL is enabled on the volume. To find out if DRL logging is active, see "Determining if DRL logging is active on a volume" on page 278.

Determining if DRL logging is active on a volume

To determine if DRL logging is active on a mirrored volume

1. Use the following `vxprint` commands to discover the name of the volume's DCO volume:
   ```
   # DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
   # DCOVOL=`vxprint [-g diskgroup] -F%parent_vol $DCONAME`
   ```
2. Use the `vxprint` command on the DCO volume to find out if DRL logging is active:
   ```
   # vxprint [-g diskgroup] -F%drllogging $DCOVOL
   ```
 This command returns `on` if DRL logging is enabled.

Disabling and re-enabling DRL

To disable DRL (configured using a version 20 DCO volume) on a volume

```
# vxvol [-g diskgroup] set drl=off volume
```

To re-enable DRL on a volume, enter this command:

```
# vxvol [-g diskgroup] set drl=on volume
```

To re-enable sequential DRL on a volume, enter:

```
# vxvol [-g diskgroup] set drl=sequential volume
```

You can use these commands to change the DRL policy on a volume by first disabling and then re-enabling DRL as required. DRL is automatically disabled if a data change map (DCM, used with Veritas Volume Replicator) is attached to a volume.

Removing support for DRL and instant snapshots from a volume

To remove support for DRL and instant snapshot operation from a volume, use the following command to remove the DCO and DCO volume that are associated with the volume:

```
# vxsnap [-g diskgroup] unprepare volume
```

This command also has the effect of disabling FastResync tracking on the volume.

Note: This command fails if the volume is part of a snapshot hierarchy.

Upgrading existing volumes to use version 20 DCOs

The procedure described in this section describes how to upgrade a volume created before VxVM 4.0 so that it can take advantage of new features such as instant snapshots, and DRL logs that are configured within the DCO volume. This requires upgrading the version of the disk groups, removing any existing snapshots and version 0 DCOs that are associated with volumes in the disk groups, and finally configuring the volumes with version 20 DCOs.

Note: The plexes of the DCO volume require persistent storage space on disk to be available. To make room for the DCO plexes, you may need to add extra disks to the disk group, or reconfigure existing volumes to free up space in the disk group. Another way to add disk space is to use the disk group move feature to bring in spare disks from a different disk group. For more information, see "Reorganizing the contents of disk groups" on page 195.

To upgrade an existing disk group and the volumes that it contains

1 Upgrade the disk group that contains the volume to the latest version before performing the remainder of the procedure described in this section. Use the following command to check the version of a disk group:

```
# vxdg list diskgroup
```

To upgrade a disk group to the latest version, use the following command:

```
# vxdg upgrade diskgroup
```

For more information, see "Upgrading a disk group" on page 208.

2 Use the following command to discover which volumes in the disk group have version 0 DCOs associated with them:

```
# vxprint [-g diskgroup] -F "%name" -e "v_hasdcolog"
```

Note: This command assumes that the volumes can only have version 0 DCOs as the disk group has just been upgraded. See "Determining the DCO version number" on page 277 for a description of how to find out the DCO version number of a volume in any disk group.

Repeat the following steps to upgrade each volume within the disk group as required.

3 If the volume to be upgraded has a traditional DRL plex or subdisk (that is, the DRL logs are not held in a version 20 DCO volume), use the following command to remove this:

```
# vxassist [-g diskgroup] remove log volume [nlog=n]
```

Use the optional attribute `nlog=`*n* to specify the number, *n*, of logs to be removed. By default, the `vxassist` command removes one log.

4 For a volume that has one or more associated snapshot volumes, use the following command to reattach and resynchronize each snapshot:

```
# vxassist [-g diskgroup] snapback snapvol
```

If FastResync was enabled on the volume before the snapshot was taken, the data in the snapshot plexes is quickly resynchronized from the original volume. If FastResync was not enabled, a full resynchronization is performed.

5 Use the following command to turn off FastResync for the volume:

```
# vxvol [-g diskgroup] set fastresync=off volume
```

6 Use the following command to dissociate a version 0 DCO object, DCO volume and snap objects from the volume:

```
# vxassist [-g diskgroup] remove log volume logtype=dco
```

7 Use the following command on the volume to upgrade it:

```
# vxsnap [-g diskgroup] prepare volume [ndcomirs=number] \
  [regionsize=size] [drl=on|sequential|off] \
  [storage_attribute ...]
```

The `ndcomirs` attribute specifies the number of DCO plexes that are created in the DCO volume. It is recommended that you configure as many DCO plexes as there are data and snapshot plexes in the volume. The DCO plexes are used to set up a DCO volume for any snapshot volume that you

subsequently create from the snapshot plexes. For example, specify `ndcomirs=5` for a volume with 3 data plexes and 2 snapshot plexes.

The value of the `regionsize` attribute specifies the size of the tracked regions in the volume. A write to a region is tracked by setting a bit in the change map. The default value is `64k` (64KB). A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

To enable DRL logging on the volume, specify `drl=on` (this is the default setting). If sequential DRL is required, specify `drl=sequential`.If DRL is not required, specify `drl=off`.

You can also specify `vxassist`-style storage attributes to define the disks that can or cannot be used for the plexes of the DCO volume.

Note: The `vxsnap prepare` command automatically enables FastResync on the volume and on any snapshots that are generated from it.

If the volume is a RAID-5 volume, it is converted to a layered volume that can be used with snapshots and FastResync.

Adding traditional DRL logging to a mirrored volume

Note: The procedure described in this section creates a DRL log that is configured within a dedicated DRL plex. The version 20 DCO volume layout includes space for a DRL log. The new DCO volume layout also supports traditional (third-mirror), instant (copy-on-write), and instant space-optimized snapshots. However, a version 20 DCO volume cannot be used in conjunction with a separate DRL plex. For full details, see “Preparing a volume for DRL and instant snapshots” on page 275.

To put dirty region logging (DRL) into effect for a mirrored volume, a log subdisk must be added to that volume. Only one log subdisk can exist per plex.

To add DRL logs to an existing volume, use the following command:

```
# vxassist [-b] [-g diskgroup] addlog volume logtype=drl \
  [nlog=n] [loglen=size]
```

Note: If specified, the `-b` option makes adding the new logs a background task.

The `nlog` attribute can be used to specify the number of log plexes to add. By default, one log plex is added. The `loglen` attribute specifies the size of the log,

where each bit represents one region in the volume. For example, the size of the log would need to be 20K for a 10GB volume with a region size of 64 kilobytes.

For example, to add a single log plex for the volume `vol03`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg addlog vol03 logtype=drl
```

When the `vxassist` command is used to add a log subdisk to a volume, by default a log plex is also created to contain the log subdisk unless you include the keyword `nolog` in the layout specification.

For a volume that will be written to sequentially, such as a database log volume, include the `logtype=drlseq` attribute to specify that sequential DRL is to be used:

```
# vxassist -g mydg addlog volume logtype=drlseq [nlog=n]
```

Once created, the plex containing a log subdisk can be treated as a regular plex. Data subdisks can be added to the log plex. The log plex and log subdisk can be removed using the procedure described in "Removing a traditional DRL log" on page 282.

Removing a traditional DRL log

Note: The procedure described in this section removes a DRL log that is configured within a dedicated DRL plex. The version 20 DCO volume layout includes space for a DRL log.

To remove a DRL log, use the `vxassist` command as follows:

```
# vxassist [-g diskgroup] remove log volume logtype=drl [nlog=n]
```

By default, the `vxassist` command removes one log. Use the optional attribute `nlog=n` to specify the number of logs that are to remain after the operation completes.

You can use storage attributes to specify the storage from which a log is to be removed. For example, to remove a log on disk `mydg10` from volume vol01, enter:

```
# vxassist -g mydg remove log vol01 !mydg10 logtype=drl
```

Adding a RAID-5 log

Note: You need a full license to use this feature.

Only one RAID-5 plex can exist per RAID-5 volume. Any additional plexes become RAID-5 log plexes, which are used to log information about data and parity being written to the volume. When a RAID-5 volume is created using the `vxassist` command, a log plex is created for that volume by default.

To add a RAID-5 log to an existing volume, use the following command:

```
# vxassist [-b] [-g diskgroup] addlog volume [loglen=length]
```

Note: If specified, the `-b` option makes adding the new log a background task.

You can specify the log length used when adding the first log to a volume. Any logs that you add subsequently are configured with the same length as the existing log.

For example, to create a log for the RAID-5 volume `volraid`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg addlog volraid
```

Adding a RAID-5 log using vxplex

As an alternative to using `vxassist`, you can add a RAID-5 log using the `vxplex` command. For example, to attach a RAID-5 log plex, `r5log`, to a RAID-5 volume, `r5vol`, in the disk group, `mydg`, use the following command:

```
# vxplex -g mydg att r5vol r5log
```

The attach operation can only proceed if the size of the new log is large enough to hold all of the data on the stripe. If the RAID-5 volume already contains logs, the new log length is the minimum of each individual log length. This is because the new log is a mirror of the old logs.

If the RAID-5 volume is not enabled, the new log is marked as BADLOG and is enabled when the volume is started. However, the contents of the log are ignored.

If the RAID-5 volume is enabled and has other enabled RAID-5 logs, the new log's contents are synchronized with the other logs.

If the RAID-5 volume currently has no enabled logs, the new log is zeroed before it is enabled.

Removing a RAID-5 log

To identify the plex of the RAID-5 log, use the following command:

```
# vxprint [-g diskgroup] -ht volume
```

where *volume* is the name of the RAID-5 volume. For a RAID-5 log, the output lists a plex with a STATE field entry of LOG.

To dissociate and remove a RAID-5 log and any associated subdisks from an existing volume, use the following command:

```
# vxplex [-g diskgroup] -o rm dis plex
```

For example, to dissociate and remove the log plex volraid-02 from volraid in the disk group, mydg, use the following command:

```
# vxplex -g mydg -o rm dis volraid-02
```

You can also remove a RAID-5 log with the vxassist command, as follows:

```
# vxassist [-g diskgroup] remove log volume [nlog=n]
```

By default, the vxassist command removes one log. Use the optional attribute nlog=*n* to specify the number of logs that are to remain after the operation completes.

Note: When removing the log leaves the volume with less than two valid logs, a warning is printed and the operation is not allowed to continue. The operation may be forced by additionally specifying the -f option to vxplex or vxassist.

Resizing a volume

Resizing a volume changes the volume size. For example, you might need to increase the length of a volume if it is no longer large enough for the amount of data to be stored on it. To resize a volume, use one of the commands: vxresize (preferred), vxassist, or vxvol. Alternatively, you can use the graphical Veritas Enterprise Administrator (VEA) to resize volumes.

If a volume is increased in size, the vxassist command automatically locates available disk space. The vxresize command requires that you specify the names of the disks to be used to increase the size of a volume. The vxvol command requires that you have previously ensured that there is sufficient space available in the plexes of the volume to increase its size. The vxassist and vxresize commands automatically free unused space for use by the disk group. For the vxvol command, you must do this yourself. To find out by how much you can grow a volume, use the following command:

```
# vxassist [-g diskgroup] maxgrow volume
```

When you resize a volume, you can specify the length of a new volume in sectors, kilobytes, megabytes, or gigabytes. The unit of measure is added as a suffix to the length (s, m, k, or g). If no unit is specified, sectors are assumed. The

vxassist command also allows you to specify an increment by which to change the volume's size.

Caution: If you use vxassist or vxvol to resize a volume, do not shrink it below the size of the file system which is located on it. If you do not shrink the file system first, you risk unrecoverable data loss. If you have a VxFS file system, shrink the file system first, and then shrink the volume. Other file systems may require you to back up your data so that you can later recreate the file system and restore its data.

Resizing volumes using vxresize

Use the vxresize command to resize a volume containing a file system. Although other commands can be used to resize volumes containing file systems, the vxresize command offers the advantage of automatically resizing certain types of file system as well as the volume.

See the following table for details of what operations are permitted and whether the file system must first be unmounted to resize the file system:

Table 8-1 Permitted resizing operations on file systems

	Online JFS (Full-VxFS)	Base JFS (Lite-VxFS)	HFS
Mounted File System	Grow and shrink	Not allowed	Not allowed
Unmounted File System	Grow only	Grow only	Grow only

For example, the following command resizes the 1-gigabyte volume, homevol, in the disk group, mydg, that contains a VxFS file system to 10 gigabytes using the spare disks mydg10 and mydg11:

```
# vxresize -g mydg -b -F vxfs -t homevolresize homevol 10g \
  mydg10 mydg11
```

The -b option specifies that this operation runs in the background. Its progress can be monitored by specifying the task tag homevolresize to the vxtask command.

Note the following restrictions for using vxresize:

- vxresize works with VxFS, JFS (derived from VxFS) and HFS file systems only.
- In some situations, when resizing large volumes, vxresize may take a long time to complete.

- Resizing a volume with a usage type other than FSGEN or RAID5 can result in loss of data. If such an operation is required, use the `-f` option to forcibly resize such a volume.
- You cannot resize a volume that contains plexes with different layout types. Attempting to do so results in the following error message:
 `VxVM vxresize ERROR V-5-1-2536 Volume volume has different organization in each mirror`
 To resize such a volume successfully, you must first reconfigure it so that each data plex has the same layout.

For more information about the `vxresize` command, see the `vxresize`(1M) manual page.

Resizing volumes using vxassist

The following modifiers are used with the `vxassist` command to resize a volume:

`growto`	Increase volume to a specified length.
`growby`	Increase volume by a specified amount.
`shrinkto`	Reduce volume to a specified length.
`shrinkby`	Reduce volume by a specified amount.

Extending to a given length

To extend a volume *to* a specific length, use the following command:

```
# vxassist [-b] [-g diskgroup] growto volume length
```

Note: If specified, the `-b` option makes growing the volume a background task.

For example, to extend `volcat` to 2000 sectors, use the following command:

```
# vxassist -g mydg growto volcat 2000
```

Note: If you previously performed a relayout on the volume, additionally specify the attribute `layout=nodiskalign` to the `growto` command if you want the subdisks to be grown using contiguous disk space.

Extending by a given length

To extend a volume *by* a specific length, use the following command:

```
# vxassist [-b] [-g diskgroup] growby volume length
```

Note: If specified, the `-b` option makes growing the volume a background task.

For example, to extend `volcat` by 100 sectors, use the following command:

```
# vxassist -g mydg growby volcat 100
```

Note: If you previously performed a relayout on the volume, additionally specify the attribute `layout=nodiskalign` to the `growby` command if you want the subdisks to be grown using contiguous disk space.

Shrinking to a given length

To shrink a volume *to* a specific length, use the following command:

```
# vxassist [-g diskgroup] shrinkto volume length
```

For example, to shrink `volcat` to 1300 sectors, use the following command:

```
# vxassist -g mydg shrinkto volcat 1300
```

Caution: Do not shrink the volume below the current size of the file system or database using the volume. The `vxassist shrinkto` command can be safely used on empty volumes.

Shrinking by a given length

To shrink a volume *by* a specific length, use the following command:

```
# vxassist [-g diskgroup] shrinkby volume length
```

For example, to shrink `volcat` by 300 sectors, use the following command:

```
# vxassist -g mydg shrinkby volcat 300
```

Caution: Do not shrink the volume below the current size of the file system or database using the volume. The `vxassist shrinkby` command can be safely used on empty volumes.

Resizing volumes using vxvol

To change the length of a volume using the `vxvol set` command, use the following command:

```
# vxvol [-g diskgroup] set len=length volume
```

For example, to change the length of the volume, `vol01`, in the disk group, `mydg`, to 100000 sectors, use the following command:

```
# vxvol -g mydg set len=100000 vol01
```

Note: The `vxvol set len` command cannot increase the size of a volume unless the needed space is available in the plexes of the volume. When the size of a volume is reduced using the `vxvol set len` command, the freed space is not released into the disk group's free space pool.

If a volume is active and its length is being reduced, the operation must be forced using the `-o force` option to `vxvol`. This prevents accidental removal of space from applications using the volume.

The length of logs can also be changed using the following command:

```
# vxvol [-g diskgroup] set loglen=length log_volume
```

Note: Sparse log plexes are not valid. They must map the entire length of the log. If increasing the log length would make any of the logs invalid, the operation is not allowed. Also, if the volume is not active and is dirty (for example, if it has not been shut down cleanly), the log length cannot be changed. This avoids the loss of any of the log contents (if the log length is decreased), or the introduction of random data into the logs (if the log length is being increased).

Setting tags on volumes

Volume tags are used to implement the Dynamic Storage Tiering feature of the Storage Foundation software. For more information about this feature, see the *Veritas File System Administrator's Guide*.

You can use the following forms of the `vxassist` command to set a named tag and optional tag value on a volume, to replace a tag, and to remove a tag from a volume:

```
# vxassist [-g diskgroup] settag volume tagname[=tagvalue]
# vxassist [-g diskgroup] replacetag volume oldtag newtag
# vxassist [-g diskgroup] removetag volume tagname
```

To list the tags that are associated with a volume, use this command:

```
# vxassist [-g diskgroup] listtag volume
```

To list the volumes that have a specified tag name, use this command:

```
# vxassist [-g diskgroup] list tag=tagname volume
```

Tag names and tag values are case-sensitive character strings of up to 256 characters. Tag names can consist of letters (A through Z and a through z), numbers (0 through 9), dashes (-), underscores (_) or periods (.) from the ASCII character set. A tag name must start with either a letter or an underscore. Tag values can consist of any character from the ASCII character set with a decimal value from 32 through 127. If a tag value includes any spaces, quote the specification to protect it from the shell, as shown here:

```
# vxassist -g mydg settag myvol "dbvol=table space 1"
```

Dotted tag hierarchies are understood by the `list` operation. For example, the listing for `tag=a.b` includes all volumes that have tag names that start with `a.b`.

The tag names `site`, `udid` and `vdid` are reserved and should not be used. To avoid possible clashes with future product features, it is recommended that tag names do not start with any of the following strings: `asl`, `be`, `isp`, `nbu`, `sf`, `symc` or `vx`.

Changing the read policy for mirrored volumes

VxVM offers the choice of the following read policies on the data plexes in a mirrored volume:

`round`	Reads each plex in turn in "round-robin" fashion for each nonsequential I/O detected. Sequential access causes only one plex to be accessed. This takes advantage of the drive or controller read-ahead caching policies.
`prefer`	Reads first from a plex that has been named as the preferred plex.
`select`	Chooses a default policy based on plex associations to the volume. If the volume has an enabled striped plex, the `select` option defaults to preferring that plex; otherwise, it defaults to round-robin.
`siteread`	Reads preferentially from plexes at the locally defined site. This is the default policy for volumes in disk groups where site consistency has been enabled. See "Administering sites and remote mirrors" on page 431.
`split`	Divides read requests and distributes them across all the available plexes.

Note: You cannot set the read policy on a RAID-5 volume. RAID-5 plexes have their own read policy (RAID).

To set the read policy to `round`, use the following command:

```
# vxvol [-g diskgroup] rdpol round volume
```

For example, to set the read policy for the volume, `vol01`, in disk group, `mydg`, to round-robin, use the following command:

```
# vxvol -g mydg rdpol round vol01
```

To set the read policy to `prefer`, use the following command:

```
# vxvol [-g diskgroup] rdpol prefer volume preferred_plex
```

For example, to set the policy for `vol01` to read preferentially from the plex `vol01-02`, use the following command:

```
# vxvol -g mydg rdpol prefer vol01 vol01-02
```

To set the read policy to `select`, use the following command:

```
# vxvol [-g diskgroup] rdpol select volume
```

For more information about how read policies affect performance, see "Volume read policies" on page 466.

Removing a volume

Once a volume is no longer necessary (it is inactive and its contents have been archived, for example), it is possible to remove the volume and free up the disk space for other uses.

To stop all activity on a volume before removing it

1. Remove all references to the volume by application programs, including shells, that are running on the system.
2. If the volume is mounted as a file system, unmount it with this command:
   ```
   # umount /dev/vx/dsk/diskgroup/volume
   ```
3. If the volume is listed in the `/etc/fstab` file, remove its entry by editing this file. Refer to your operating system documentation for more information about the format of this file and how you can modify it.
4. Stop all activity by VxVM on the volume with the command:
   ```
   # vxvol [-g diskgroup] stop volume
   ```

After following these steps, remove the volume with the `vxassist` command:

```
# vxassist [-g diskgroup] remove volume volume
```

Alternatively, you can use the `vxedit` command to remove a volume:

```
# vxedit [-g diskgroup] [-r] [-f] rm volume
```

The `-r` option to `vxedit` indicates recursive removal. This removes all plexes associated with the volume and all subdisks associated with those plexes. The `-f` option to `vxedit` forces removal. This is necessary if the volume is still enabled.

Moving volumes from a VM disk

Before you disable or remove a disk, you can move the data from that disk to other disks on the system that have sufficient space.

To move volumes from a disk

1 Select menu item `6 (Move volumes from a disk)` from the `vxdiskadm` main menu.

2 At the following prompt, enter the disk name of the disk whose volumes you wish to move, as follows:

```
Move volumes from a disk
Menu: VolumeManager/Disk/Evacuate
Use this menu operation to move any volumes that are using a
disk onto other disks. Use this menu immediately prior to
removing a disk, either permanently or for replacement. You
can specify a list of disks to move volumes onto, or you can
move the volumes to any available disk space in the same disk
group.

NOTE: Simply moving volumes off of a disk, without also
removing the disk, does not prevent volumes from being moved
onto the disk by future operations. For example, using two
consecutive move operations may move volumes from the
second disk to the first.

Enter disk name [<disk>,list,q,?] mydg01
```

You can now optionally specify a list of disks to which the volume(s) should be moved:

```
VxVM INFO V-5-2-516 You can now specify a list of disks to move
onto.  Specify a list of disk media names (e.g., mydg01) all on
one line separated by blanks. If you do not enter any disk
media names, then the volumes will be moved to any available
space in the disk group.
```

At the prompt, press Return to move the volumes onto available space in the disk group, or specify the disks in the disk group that should be used:

```
Enter disks [<disk ...>,list]
VxVM  NOTICE V-5-2-283 Requested operation is to move all
volumes from disk mydg01 in group mydg.

NOTE: This operation can take a long time to complete.

Continue with operation? [y,n,q,?] (default: y)
```

As the volumes are moved from the disk, the `vxdiskadm` program displays the status of the operation:

```
VxVM vxevac INFO V-5-2-24   Move volume voltest ...
```

When the volumes have all been moved, the `vxdiskadm` program displays the following success message:

```
VxVM  INFO V-5-2-188 Evacuation of disk mydg02 is complete.
```

3 At the following prompt, indicate whether you want to move volumes from another disk (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Move volumes from another disk? [y,n,q,?] (default: n)
```

Enabling FastResync on a volume

Note: The recommended method for enabling FastResync on a volume with a version 20 DCO is to use the `vxsnap prepare` command as described in "Preparing a volume for DRL and instant snapshots" on page 275.

You need a Veritas FlashSnap™ or FastResync license to use this feature.

FastResync performs quick and efficient resynchronization of stale mirrors. It also increases the efficiency of the VxVM snapshot mechanism when used with operations such as backup and decision support. See "Administering volume snapshots" on page 303 and "FastResync" on page 66 for more information.

There are two possible versions of FastResync that can be enabled on a volume:

- Persistent FastResync holds copies of the FastResync maps on disk. These can be used for the speedy recovery of mirrored volumes if a system is rebooted. This form of FastResync requires that either a version 0 or a version 20 data change object (DCO) and DCO volume first be associated with the volume.
 See "Adding a version 0 DCO and DCO volume" on page 356, and "Preparing a volume for DRL and instant snapshots" on page 275 for more information on version 0 and version 20 DCO volumes respectively.
 If the existing volume was created in a previous release of VxVM, and it has any attached snapshot plexes or it is associated with any snapshot volumes, follow the procedure given in "Upgrading existing volumes to use version 20 DCOs" on page 279.
- Non-Persistent FastResync holds the FastResync maps in memory. These do not survive on a system that is rebooted.

By default, FastResync is not enabled on newly created volumes. Specify the `fastresync=on` attribute to the `vxassist make` command if you want to enable FastResync on a volume that you are creating.

Note: It is not possible to configure both Persistent and Non-Persistent FastResync on a volume. Persistent FastResync is used if a DCO is associated with the volume. Otherwise, Non-Persistent FastResync is used.

To turn FastResync on for an existing volume, specify `fastresync=on` to the `vxvol` command as shown here:

```
# vxvol [-g diskgroup] set fastresync=on volume
```

Note: To use FastResync with a snapshot, FastResync must be enabled before the snapshot is taken, and must remain enabled until after the snapback is completed.

Checking whether FastResync is enabled on a volume

To check whether FastResync is enabled on a volume, use the following command:

```
# vxprint [-g diskgroup] -F%fastresync volume
```

This command returns `on` if FastResync is enabled; otherwise, it returns `off`.

If FastResync is enabled, to check whether it is Non-Persistent or Persistent FastResync, use the following command:

```
# vxprint [-g diskgroup] -F%hasdcolog volume
```

This command returns `on` if Persistent FastResync is enabled; otherwise, it returns `off`.

To list all volumes on which Non-Persistent FastResync is enabled, use the following command:

```
# vxprint [-g diskgroup] -F "%name" \
  -e "v_fastresync=on && !v_hasdcolog"
```

To list all volumes on which Persistent FastResync is enabled, use the following command:

```
# vxprint [-g diskgroup] -F "%name" -e "v_fastresync=on \
  && v_hasdcolog"
```

Disabling FastResync

Use the `vxvol` command to turn off Persistent or Non-Persistent FastResync for an existing volume, as shown here:

```
# vxvol [-g diskgroup] set fastresync=off volume
```

Turning FastResync off releases all tracking maps for the specified volume. All subsequent reattaches will not use the FastResync facility, but perform a full resynchronization of the volume. This occurs even if FastResync is later turned on.

Performing online relayout

Note: You need a full license to use this feature.

You can use the `vxassist relayout` command to reconfigure the layout of a volume without taking it offline. The general form of this command is:

```
# vxassist [-b] [-g diskgroup] relayout volume [layout=layout] \
  [relayout_options]
```

Note: If specified, the `-b` option makes relayout of the volume a background task.

The following are valid destination layout configurations as determined by the tables in "Permitted relayout transformations" on page 295:

`concat-mirror`—concatenated-mirror

`concat` or `span`, `nostripe`, `nomirror`—concatenated

`raid5`—RAID-5 (not supported for shared disk groups)

`stripe`—striped

`stripe-mirror`—striped-mirror

For example, the following command changes a concatenated volume, `vol02`, in disk group, `mydg`, to a striped volume with the default number of columns, 2, and default stripe unit size, 64 kilobytes:

```
# vxassist -g mydg relayout vol02 layout=stripe
```

On occasions, it may be necessary to perform a relayout on a plex rather than on a volume. See "Specifying a plex for relayout" on page 298 for more information.

Permitted relayout transformations

The tables below give details of the relayout operations that are possible for each type of source storage layout.

Table 8-2 Supported relayout transformations for concatenated volumes

Relayout to	From concat
concat	No.
concat-mirror	No. Add a mirror, and then use `vxassist convert` instead.
mirror-concat	No. Add a mirror instead.
mirror-stripe	No. Use `vxassist convert` after relayout to striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be defined.
stripe	Yes. The stripe width and number of columns may be defined.
stripe-mirror	Yes. The stripe width and number of columns may be defined.

Table 8-3 Supported relayout transformations for concatenated-mirror volumes

Relayout to	From concat-mirror
concat	No. Use `vxassist convert`, and then remove unwanted mirrors from the resulting mirrored-concatenated volume instead.
concat-mirror	No.
mirror-concat	No. Use `vxassist convert` instead.
mirror-stripe	No. Use `vxassist convert` after relayout to striped-mirror volume instead.
raid5	Yes.
stripe	Yes. This removes a mirror and adds striping. The stripe width and number of columns may be defined.
stripe-mirror	Yes. The stripe width and number of columns may be defined.

Table 8-4 Supported relayout transformations for RAID-5 volumes

Relayout to	From raid5
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be changed.
stripe	Yes. The stripe width and number of columns may be changed.
stripe-mirror	Yes. The stripe width and number of columns may be changed.

Table 8-5 Supported relayout transformations for mirrored-concatenated volumes

Relayout to	From mirror-concat
concat	No. Remove unwanted mirrors instead.
concat-mirror	No. Use `vxassist convert` instead.
mirror-concat	No.
mirror-stripe	No. Use `vxassist convert` after relayout to striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be defined. Choose a plex in the existing mirrored volume on which to perform the relayout. The other plexes are removed at the end of the relayout operation.
stripe	Yes.
stripe-mirror	Yes.

Table 8-6 Supported relayout transformations for mirrored-stripe volumes

Relayout to	From mirror-stripe
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be changed.
stripe	Yes. The stripe width or number of columns must be changed.
stripe-mirror	Yes. The stripe width or number of columns must be changed. Otherwise, use `vxassist convert`.

Table 8-7 Supported relayout transformations for unmirrored stripe and layered striped-mirror volumes

Relayout to	From stripe or stripe-mirror
concat	Yes.
concat-mirror	Yes.
mirror-concat	No. Use `vxassist convert` after relayout to concatenated-mirror volume instead.
mirror-stripe	No. Use `vxassist convert` after relayout to striped-mirror volume instead.
raid5	Yes. The stripe width and number of columns may be changed.
stripe	Yes. The stripe width or number of columns must be changed.
stripe-mirror	Yes. The stripe width or number of columns must be changed.

Specifying a non-default layout

You can specify one or more relayout options to change the default layout configuration. Examples of these options are:

ncol=*number* Specifies the number of columns.

ncol=+*number* Specifies the number of columns to add.

ncol=-*number* Specifies the number of colums to remove.

stripeunit=*size* Specifies the stripe width.

See the `vxassist`(1M) manual page for more information about relayout options.

The following are some examples of using vxassist to change the stripe width and number of columns for a striped volume in the disk group `dbaseg`:

```
# vxassist -g dbaseg relayout vol03 stripeunit=64k ncol=6
# vxassist -g dbaseg relayout vol03 ncol=+2
# vxassist -g dbaseg relayout vol03 stripeunit=128k
```

The next example changes a concatenated volume to a RAID-5 volume with four columns:

```
# vxassist -g fsgrp relayout vol04 layout=raid5 ncol=4
```

Specifying a plex for relayout

Any layout can be changed to RAID-5 if there are sufficient disks and space in the disk group. If you convert a mirrored volume to RAID-5, you must specify which plex is to be converted. All other plexes are removed when the conversion has finished, releasing their space for other purposes. If you convert a mirrored volume to a layout other than RAID-5, the unconverted plexes are not removed. You can specify the plex to be converted by naming it in place of a volume:

```
# vxassist [-g diskgroup] relayout plex [layout=layout] \
  [relayout_options]
```

Tagging a relayout operation

If you want to control the progress of a relayout operation, for example to pause or reverse it, use the `-t` option to `vxassist` to specify a task tag for the operation. For example, this relayout is performed as a background task and has the tag `myconv`:

```
# vxassist -b -g fsgrp -t myconv relayout vol04 layout=raid5 \
  ncol=4
```

See the following sections, “Viewing the status of a relayout” on page 299 and “Controlling the progress of a relayout” on page 299, for more information about tracking and controlling the progress of relayout.

Viewing the status of a relayout

Online relayout operations take some time to perform. You can use the `vxrelayout` command to obtain information about the status of a relayout operation. For example, the command:

```
# vxrelayout -g mydg status vol04
```

might display output similar to this:

```
STRIPED, columns=5, stwidth=128--> STRIPED, columns=6,
stwidth=128
Relayout running, 68.58% completed.
```

In this example, the reconfiguration of a striped volume from 5 to 6 columns is in progress, and is just over two-thirds complete.

See the `vxrelayout`(1M) manual page for more information about this command.

If you specified a task tag to `vxassist` when you started the relayout, you can use this tag with the `vxtask` command to monitor the progress of the relayout. For example, to monitor the task tagged as `myconv`, enter:

```
# vxtask monitor myconv
```

Controlling the progress of a relayout

You can use the `vxtask` command to stop (`pause`) the relayout temporarily, or to cancel it altogether (`abort`). If you specified a task tag to `vxassist` when you started the relayout, you can use this tag to specify the task to `vxtask`. For example, to pause the relayout operation tagged as `myconv`, enter:

```
# vxtask pause myconv
```

To resume the operation, use the `vxtask` command:

```
# vxtask resume myconv
```

For relayout operations that have not been stopped using the `vxtask pause` command (for example, the `vxtask abort` command was used to stop the task, the transformation process died, or there was an I/O failure), resume the relayout by specifying the `start` keyword to `vxrelayout`, as shown here:

```
# vxrelayout -g mydg -o bg start vol04
```

Note: If you use the `vxrelayout start` command to restart a relayout that you previously suspended using the `vxtask pause` command, a new untagged task is created to complete the operation. You cannot then use the original task tag to control the relayout.

The `-o bg` option restarts the relayout in the background. You can also specify the `slow` and `iosize` option modifiers to control the speed of the relayout and the size of each region that is copied. For example, the following command

inserts a delay of 1000 milliseconds (1 second) between copying each 10-megabyte region:

```
# vxrelayout -g mydg -o bg,slow=1000,iosize=10m start vol04
```

The default delay and region size values are 250 milliseconds and 1 megabyte respectively.

To reverse the direction of relayout operation that is currently stopped, specify the `reverse` keyword to `vxrelayout` as shown in this example:

```
# vxrelayout -g mydg -o bg reverse vol04
```

This undoes changes made to the volume so far, and returns it to its original layout.

If you cancel a relayout using `vxtask abort`, the direction of the conversion is also reversed, and the volume is returned to its original configuration.

See the `vxrelayout`(1M) and vxtask(1M) manual pages for more information about these commands. See "Managing tasks with vxtask" on page 268 for more information about controlling tasks in VxVM.

Converting between layered and non-layered volumes

The `vxassist convert` command transforms volume layouts between layered and non-layered forms:

```
# vxassist [-b] [-g diskgroup] convert volume [layout=layout] \
  [convert_options]
```

Note: If specified, the `-b` option makes conversion of the volume a background task.

The following conversion layouts are supported:

`stripe-mirror`	mirrored-stripe to striped-mirror
`mirror-stripe`	striped-mirror to mirrored-stripe
`concat-mirror`	mirrored-concatenated to concatenated-mirror
`mirror-concat`	concatenated-mirror to mirrored-concatenated

Volume conversion can be used before or after performing online relayout to achieve a larger number of transformations than would otherwise be possible. During relayout process, a volume may also be converted into a layout that is intermediate to the one that is desired. For example, to convert a volume from a 4-column mirrored-stripe to a 5-column mirrored-stripe, first use `vxassist relayout` to convert the volume to a 5-column striped-mirror as shown here:

```
# vxassist -g mydg relayout vol1 ncol=5
```

When the relayout has completed, use the `vxassist convert` command to change the resulting layered striped-mirror volume to a non-layered mirrored-stripe:

```
# vxassist -g mydg convert vol1 layout=mirror-stripe
```

Note: If the system crashes during relayout or conversion, the process continues when the system is rebooted. However, if the crash occurred during the first stage of a two-stage relayout and convert operation, only the first stage will be completed. You must run `vxassist convert` manually to complete the operation.

Chapter 9

Administering volume snapshots

Veritas Volume Manager (VxVM) provides the capability for taking an image of a volume at a given point in time. Such an image is referred to as a *volume snapshot*. You can also take a snapshot of a volume set as described in "Creating instant snapshots of volume sets" on page 334.

Volume snapshots allow you to make backup copies of your volumes online with minimal interruption to users. You can then use the backup copies to restore data that has been lost due to disk failure, software errors or human mistakes, or to create replica volumes for the purposes of report generation, application development, or testing.

For an introduction to the volume snapshot feature, see "Volume snapshots" on page 63. More detailed descriptions of each type of volume snapshot and the operations that you can perform on them may be found in the following sections:

- "Traditional third-mirror break-off snapshots" on page 305
- "Full-sized instant snapshots" on page 307
- "Space-optimized instant snapshots" on page 309
- "Emulation of third-mirror break-off snapshots" on page 310
- "Linked break-off snapshot volumes" on page 311
- "Cascaded snapshots" on page 312
- "Creating multiple snapshots" on page 317
- "Restoring the original volume from a snapshot" on page 317

Note: A volume snapshot represents the data that exists in a volume at a given point in time. As such, VxVM does not have any knowledge of data that is cached by the overlying file system, or by applications such as databases that have files open in the file system. If the `fsgen` volume usage type is set on a volume that contains a Veritas File System (VxFS), intent logging of the file system metadata ensures the internal consistency of the file system that is backed up. For other file system types, depending on the intent logging capabilities of the file system, there may potentially be inconsistencies between data in memory and in the snapshot image.

For databases, a suitable mechanism must additionally be used to ensure the integrity of tablespace data when the volume snapshot is taken. The facility to temporarily suspend file system I/O is provided by most modern database software. For ordinary files in a file system, which may be open to a wide variety of different applications, there may be no way to ensure the complete integrity of the file data other than by shutting down the applications and temporarily unmounting the file system. In many cases, it may only be important to ensure the integrity of file data that is not in active use at the time that you take the snapshot.

Methods of creating volume snapshots are described in the following sections:

- "Creating instant snapshots" on page 319 describes how to use the `vxsnap` command to create and administer full-sized and space-optimized instant snapshots.
- "Creating traditional third-mirror break-off snapshots" on page 348 describes how to use the `vxassist` command to create and administer traditional third-mirror snapshots.

For details of how to use volume snapshots to implement off-host online backup, see "Configuring off-host processing" on page 369.

Note: Snapshot creation using the `vxsnap` command is the preferred mechanism for implementing online and off-host point-in-time copy solutions in VxVM. Support for traditional third-mirror snapshots that are created using the `vxassist` command may be removed in a future release.

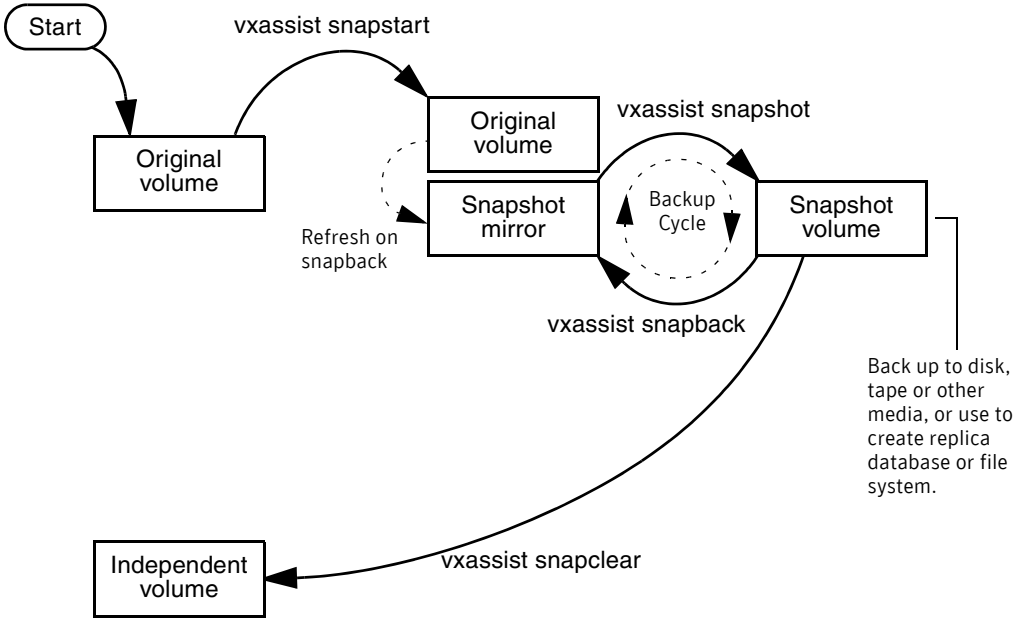
Most VxVM commands require superuser or equivalent privileges.

Full details of how to recover from failures of instant snapshot commands may be found in the "Recovery from failure of instant snapshot operations'' chapter of the *Veritas Volume Manager Troubleshooting Guide*.

Traditional third-mirror break-off snapshots

The traditional *third-mirror break-off* volume snapshot model that is supported by the `vxassist` command is shown in Figure 9-1. This also shows the transitions that are supported by the `snapback` and `snapclear` commands to `vxassist`.

Figure 9-1 Third-mirror snapshot creation and usage



The `vxassist snapstart` command creates a mirror to be used for the snapshot, and attaches it to the volume as a snapshot mirror. (The `vxassist snapabort` command can be used to cancel this operation and remove the snapshot mirror.)

Note: As is usual when creating a mirror, the process of copying the volume's contents to the new snapshot plexes can take some time to complete. For methods of making snapshot plexes immediately available, see "Full-sized instant snapshots" on page 307 and "Space-optimized instant snapshots" on page 309.

When the attachment is complete, the `vxassist snapshot` command is used to create a new snapshot volume by taking one or more snapshot mirrors to use as

its data plexes. The snapshot volume contains a copy of the original volume's data at the time that you took the snapshot. If more than one snapshot mirror is used, the snapshot volume is itself mirrored.

The command, `vxassist snapback`, can be used to return snapshot plexes to the original volume from which they were snapped, and to resynchronize the data in the snapshot mirrors from the data in the original volume. This enables you to refresh the data in a snapshot after each time that you use it to make a backup. You can also use a variation of the same command to restore the contents of the original volume from a snapshot that you took at an earlier point in time. See "Restoring the original volume from a snapshot" on page 317 for more information.

As described in "FastResync" on page 66, you can use the FastResync feature of VxVM to minimize the time needed to resynchronize the data in the snapshot mirror. If FastResync is not enabled, a full resynchronization of the data is required.

Finally, you can use the `vxassist snapclear` command to break the association between the original volume and the snapshot volume. The snapshot volume then has an existence that is independent of the original volume. This is useful for applications that do not require the snapshot to be resynchronized with the original volume.

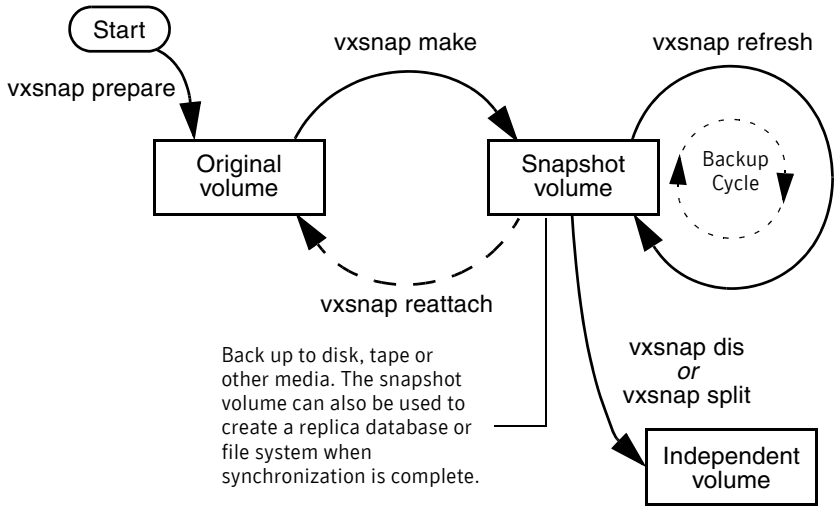
Note: The use of the `vxassist` command to administer traditional (third-mirror break-off) snapshots is not supported for volumes that are prepared for instant snapshot creation. Instead, the `vxsnap` command may be used as described in the following section.

See "Creating traditional third-mirror break-off snapshots" on page 348 for a description of the procedures for creating and using this type of snapshot.

Full-sized instant snapshots

Full-sized instant snapshots are a variation on the third-mirror volume snapshot model that make a snapshot volume available for access as soon as the snapshot plexes have been created. The full-sized instant volume snapshot model is illustrated in Figure 9-2.

Figure 9-2 Full-sized instant snapshot creation and usage in a backup cycle



To create an instant snapshot, you use the `vxsnap make` command. This command can either be applied to a suitably prepared empty volume that is to be used as the snapshot volume, or it can be used to break off one or more synchronized plexes from the original volume (which is similar to the way that the `vxassist` command creates its snapshots).

Unlike a third-mirror break-off snapshot created using the `vxassist` command, you can make a backup of a full-sized instant snapshot, instantly refresh its contents from the original volume, or attach its plexes to the original volume, without needing to completely synchronize the snapshot plexes from the original volume.

VxVM uses a *copy-on-write* mechanism to ensure that the snapshot volume preserves the contents of the original volume at the time that the snapshot is taken. Any time that the original contents of the volume are about to be overwritten, the original data in the volume is preserved on the snapshot volume before the write proceeds. As time goes by, and the contents of the

volume are updated, its original contents are gradually relocated to the snapshot volume.

If desired, you can additionally select to perform either a background (non-blocking) or foreground (blocking) synchronization of the snapshot volume. This is useful if you intend to move the snapshot volume into a separate disk group for off-host processing, or you want to use the `vxsnap dis` or `vxsnap split` commands to turn the snapshot volume into an independent volume.

The `vxsnap refresh` command allows you to update the data in a snapshot each time that you make a backup.

The command, `vxsnap reattach`, can be used to attach snapshot plexes to the original volume, and to resynchronize the data in these plexes from the original volume. Alternatively, you can use the `vxsnap restore` command to restore the contents of the original volume from a snapshot that you took at an earlier point in time. You can also choose whether or not to keep the snapshot volume after restoration of the original volume is complete. See "Restoring the original volume from a snapshot" on page 317 for more information.

By default, the FastResync feature of VxVM is used to minimize the time needed to resynchronize the data in the snapshot mirror. If FastResync is not enabled, a full resynchronization of the data is required. For details, see "FastResync" on page 66.

See "Creating and managing full-sized instant snapshots" on page 327 for details of the procedures for creating and using this type of snapshot.

For information about how to prepare an empty volume for use by full-sized instant snapshots, see "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 323.

Space-optimized instant snapshots

Volume snapshots, such as those described in "Traditional third-mirror break-off snapshots" on page 305 and "Full-sized instant snapshots" on page 307, require the creation of a complete copy of the original volume, and use as much storage space as the original volume.

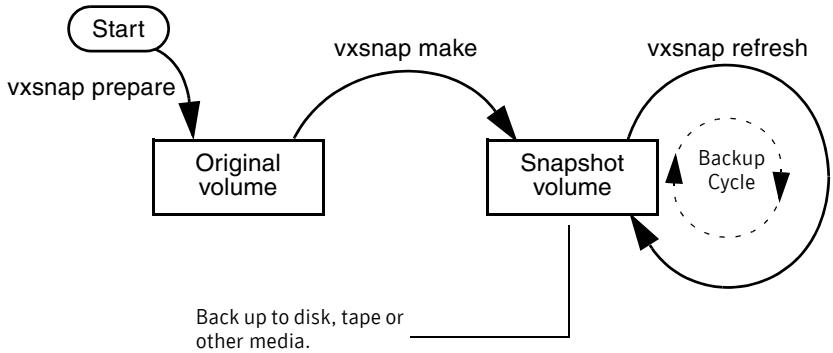
Instead of requiring a complete copy of the original volume's storage space, *space-optimized* instant snapshots use a storage cache. The size of this cache may be configured when the snapshot is created.

Note: A storage cache may be named and shared among several volumes in the same disk group. If so, the size of the cache that is declared must be the same for each volume's space-optimized snapshot. You may find it convenient to configure a single storage cache in a disk group that can be shared by all the volumes in that disk group. See "Creating a shared cache object" on page 322 for details.

When the original volume is written to, VxVM preserves the original data contents in the cache before the write is committed. As the storage cache can be configured to require much less storage than the original volume, it is referred to as being *space-optimized*. If the cache becomes too full, you can configure VxVM to grow the size of the cache automatically using any available free space in the disk group.

The instant space-optimized snapshot model is illustrated in Figure 9-3.

Figure 9-3 Space-optimized instant snapshot creation and usage in a backup cycle



As for instant snapshots, space-optimized snapshots use a copy-on-write mechanism to make them immediately available for use when they are first created, or when their data is refreshed. Unlike instant snapshots, however, you cannot enable synchronization on space-optimized snapshots, reattach them to their original volume, or turn them into independent volumes.

See "Creating and managing space-optimized instant snapshots" on page 324 for details of the procedures for creating and using this type of snapshot.

For information about how to set up a cache for use by space-optimized instant snapshots, see "Creating a shared cache object" on page 322.

Emulation of third-mirror break-off snapshots

Third-mirror break-off snapshots are suitable for write-intensive volumes (such as for database redo logs) where the copy-on-write mechanism of space-optimized or full-sized instant snapshots might degrade the performance of the volume.

If you use the `vxsnap prepare` command to enable a volume for use with instant and space-optimized snapshots, you cannot use the `vxassist` snapshot commands to administer snapshots that you create for the volume. If you require snapshots that behave as third-mirror break-off snapshots (that is, they must be fully synchronized before they can be used), there are three ways to achieve this:

- Use the `vxsnap addmir` command to create and attach one or more snapshot mirrors to the volume. When the plexes have been synchronized and are in the `SNAPDONE` state, the `vxsnap make` command can then be used with the `nmirror` attribute to create the snapshot volume. This technique is similar to using the `vxassist snapstart` and `vxassist snapshot` commands that are described in "Traditional third-mirror break-off snapshots" on page 305.
- Use the `vxsnap make` command with the `plex` attribute to use one or more existing plexes of a volume as snapshot plexes. The volume must have a sufficient number of available plexes that are in the `ACTIVE` state.

 Note: The volume must be a non-layered volume with a `mirror` or `mirror-stripe` layout, or a RAID-5 volume that you have converted to a special layered volume (see "Using a DCO and DCO volume with a RAID-5 volume" on page 277) and then mirrored.

 The plexes in a volume with a `stripe-mirror` layout are mirrored at the sub-volume level, and cannot be broken off.

- Use the `vxsnap make` command with the `sync=yes` and `type=full` attributes specified to create the snapshot volume, and then use the `vxsnap syncwait` command to wait for synchronization of the snapshot volume to complete.

See "Creating and managing third-mirror break-off snapshots" on page 329 for details of the procedures for creating and using this type of snapshot.

For information about how to add snapshot mirrors to a volume, see "Adding snapshot mirrors to a volume" on page 336.

Linked break-off snapshot volumes

A variant of the third-mirror break-off snapshot type are linked break-off snapshot volumes, which use the `vxsnap addmir` command to link a specially prepared volume with the data volume. The volume that is used for the snapshot is prepared in the same way as for full-sized instant snapshots. However, unlike full-sized instant snapshots, this volume can be set up in a different disk group from the data volume. This makes linked break-off snapshots especially suitable for off-host processing applications where you may want to create the snapshot on storage with different characteristics from that used for the data volumes. As for third-mirror break-off snapshots, you must wait for the contents of the snapshot volume to be synchronized with the data volume before you can use the `vxsnap make` command to take the snapshot.

When a link is created between a volume and the mirror that will become the snapshot, separate link objects (similar to snap objects) are associated with the volume and with its mirror. The link object for the original volume points to the mirror volume, and the link object for the mirror volume points to the original volume. All I/O is directed to both the original volume and its mirror, and a synchronization of the mirror from the data in the original volume is started.

You can use the `vxprint` command to display the state of link objects, which appear as type `ln`. Link objects can have the following states:

`ACTIVE` The mirror volume has been fully synchronized from the original volume. The `vxsnap make` command can be run to create a snapshot volume.

`ATTACHING` Synchronization of the mirror volume is in progress. The `vxsnap make` command cannot be used to create a snapshot volume until the state changes to `ACTIVE`. The `vxsnap snapwait` command can be used to wait for the synchronization to complete.

`BROKEN` The mirror volume has been detached from the original volume because of an I/O error or an unsuccessful attempt to grow the mirror volume. The `vxrecover` command can be used

to recover the mirror volume in the same way as for a DISABLED volume. See "Starting a volume" on page 271.

If you resize (that is, grow or shrink) a volume, all its ACTIVE linked mirror volumes are also resized at the same time. The volume and its mirrors can be in the same disk group or in different disk groups. If the operation is successful, the volume and its mirrors will have the same size.

If a volume has been grown, a resynchronization of the grown regions in its linked mirror volumes is started, and the links remain in the ATTACHING state until resynchronization is complete. The vxsnap snapwait command can be used to wait for the state to become ACTIVE.

When you use the vxsnap make command to create the snapshot volume, this removes the link, and establishes a snapshot relationship between the snapshot volume and the original volume.

The vxsnap reattach operation re-establishes the link relationship between the two volumes, and starts a resynchronization of the mirror volume.

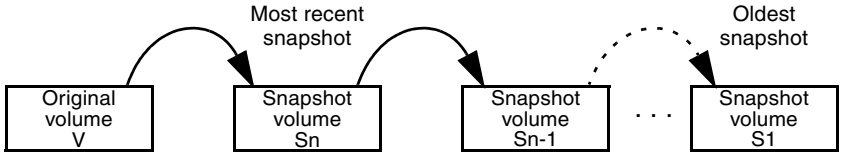
See "Creating and managing linked break-off snapshot volumes" on page 331 for details of the procedures for creating and using this type of snapshot.

For information about how to prepare an empty volume for use by full-sized instant snapshots, see "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 323.

Cascaded snapshots

A snapshot hierarchy known as a *snapshot cascade* can improve write performance for some applications. Instead of having several independent snapshots of the volume, it is more efficient to make the older snapshots into children of the latest snapshot as shown in Figure 9-4.

Figure 9-4 Snapshot cascade



A snapshot may be added to a cascade by specifying the infrontof attribute to the vxsnap make command when the second and subsequent snapshots in the cascade are created. Changes to blocks in the original volume are only written to the most recently created snapshot volume in the cascade. If an attempt is made

to read data from an older snapshot that does not exist in that snapshot, it is obtained by searching recursively up the hierarchy of more recent snapshots.

A snapshot cascade is most likely to be used for regular online backup of a volume where space-optimized snapshots are written to disk but not to tape.

A snapshot cascade improves write performance over the alternative of several independent snapshots, and also requires less disk space if the snapshots are space-optimized. Only the latest snapshot needs to be updated when the original volume is updated. If and when required, the older snapshots can obtain the changed data from the most recent snapshot.

The following points determine whether it is appropriate for an application to use a snapshot cascade:

- Deletion of a snapshot in the cascade takes time to copy the snapshot's data to the next snapshot in the cascade.
- The reliability of a snapshot in the cascade depends on all the newer snapshots in the chain. Thus the oldest snapshot in the cascade is the most vulnerable.
- Reading from a snapshot in the cascade may require data to be fetched from one or more other snapshots in the cascade.

For these reasons, it is recommended that you do not attempt to use a snapshot cascade with applications that need to remove or split snapshots from the cascade. In such cases, it may be more appropriate to create a snapshot of a snapshot as described in the following section.

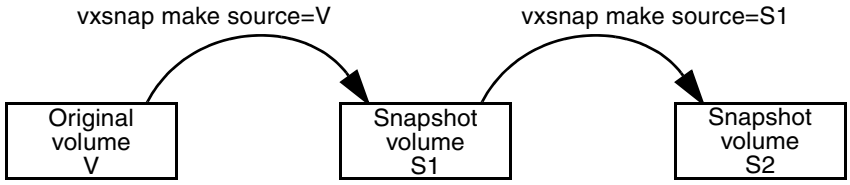
See "Adding a snapshot to a cascaded snapshot hierarchy" on page 337 for an example of the use of the `infrontof` attribute.

Note: Only unsynchronized full-sized or space-optimized instant snapshots are usually cascaded. It is of little utility to create cascaded snapshots if the `infrontof` snapshot volume is fully synchronized (as, for example, with break-off type snapshots).

Creating a snapshot of a snapshot

For some applications, it may be desirable to create a snapshot of an existing snapshot as illustrated in Figure 9-5.

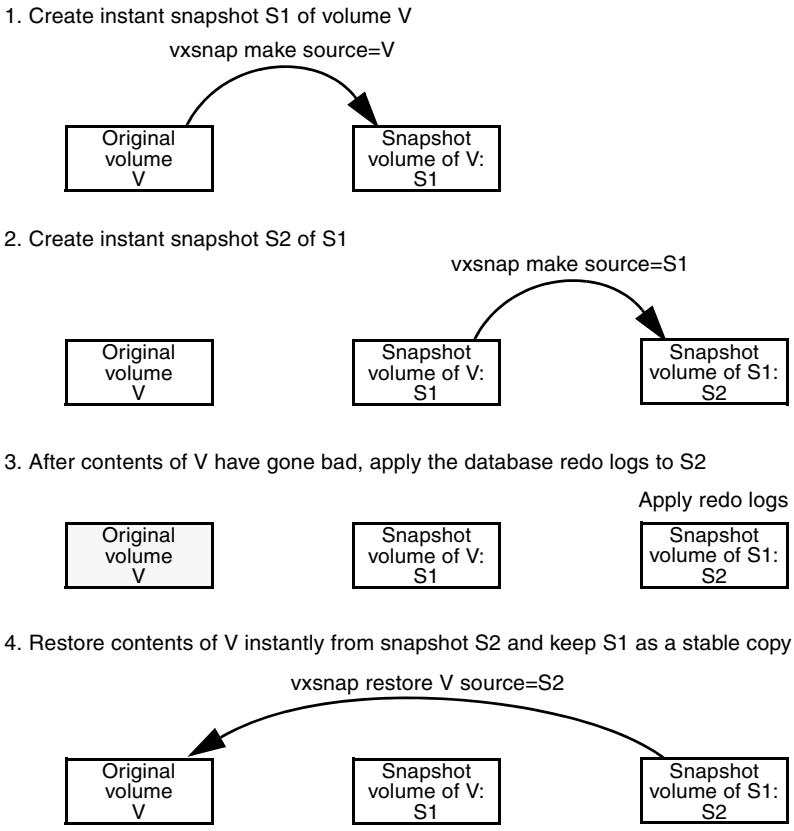
Figure 9-5 Creating a snapshot of a snapshot



Even though the arrangement of the snapshots in this figure appears similar to the snapshot hierarchy shown in "Snapshot cascade" on page 312, the relationship between the snapshots is not recursive. When reading from the snapshot `S2`, data is obtained directly from the original volume, `V`, if it does not exist in `S2` itself.

Such an arrangement may be useful if the snapshot volume, `S1`, is critical to the operation. For example, `S1` could be used as a stable copy of the original volume, V. The additional snapshot volume, `S2`, can be used to restore the original volume if that volume becomes corrupted. For a database, you might need to replay a redo log on `S2` before you could use it to restore `V`. These steps are illustrated in Figure 9-6.

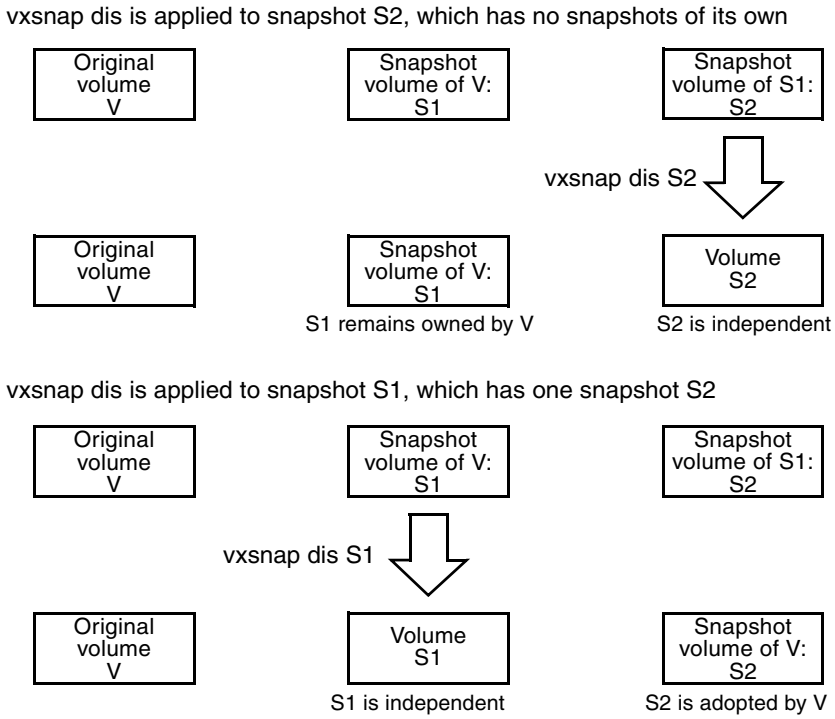
Figure 9-6 Using a snapshot of a snapshot to restore a database



If you have configured snapshots in this way, you may wish to make one or more of the snapshots into independent volumes. There are two `vxsnap` commands that you can use to do this:

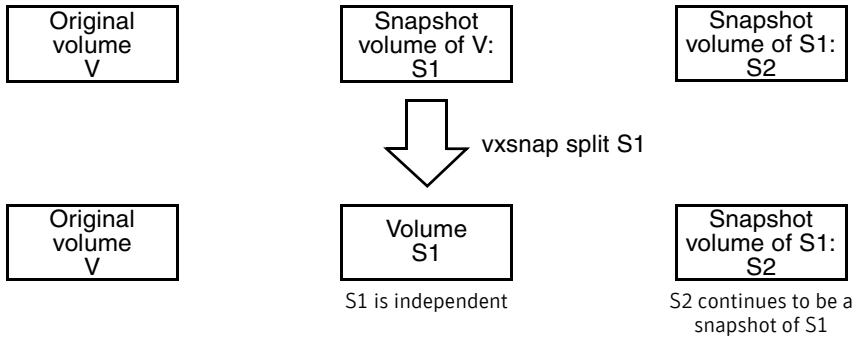
- `vxsnap dis` dissociates a snapshot volume and turns it into an independent volume. The volume to be dissociated must have been fully synchronized from its parent. If a snapshot volume has a child snapshot volume, the child must also have been fully synchronized. If the command succeeds, the child snapshot becomes a snapshot of the original volume. Figure 9-7 illustrates the effect of applying this command to snapshots with and without dependent snapshots.

Figure 9-7 Dissociating a snapshot volume



- `vxsnap split` dissociates a snapshot and its dependent snapshots from its parent volume. The snapshot volume that is to be split must have been fully synchronized from its parent volume. This operation is illustrated in Figure 9-8.

Figure 9-8 Splitting snapshots



Creating multiple snapshots

To make it easier to create snapshots of several volumes at the same time, both the `vxsnap make` and `vxassist snapshot` commands accept more than one volume name as their argument.

For traditional snapshots, you can create snapshots of all the volumes in a single disk group by specifying the option `-o allvols` to the `vxassist snapshot` command.

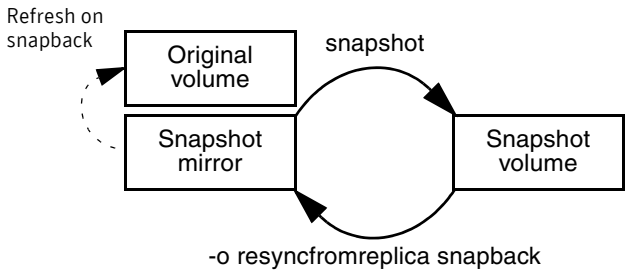
By default, each replica volume is named `SNAP`*number*`-`*volume*, where number is a unique serial number, and *volume* is the name of the volume for which a snapshot is being taken. This default can be overridden by using the option `-o name=`*pattern*, as described on the `vxsnap`(1M) and `vxassist`(1M) manual pages.

It is also possible to take several snapshots of the same volume. A new FastResync change map is produced for each snapshot taken to minimize the resynchronization time for each snapshot.

Restoring the original volume from a snapshot

For traditional snapshots, the snapshot plex is resynchronized from the data in the original volume during a `vxassist snapback` operation. Alternatively, you can choose the snapshot plex as the preferred copy of the data when performing a snapback as illustrated in Figure 9-9. Specifying the option `-o resyncfromreplica` to `vxassist` resynchronizes the original volume from the data in the snapshot.

Figure 9-9 Resynchronizing an original volume from a snapshot



Note: The original volume must not be in use during a `snapback` operation that specifies the option `-o resyncfromreplica` to resynchronize the volume from a snapshot. Stop any application, such as a database, and unmount any file systems that are configured to use the volume.

For instant snapshots, the `vxsnap restore` command may be used to restore the contents of the original volume from an instant snapshot or from a volume derived from an instant snapshot. The volume that is used to restore the original volume can either be a true backup of the contents of the original volume at some point in time, or it may have been modified in some way (for example, by applying a database log replay or by running a file system checking utility such as `fsck`) to create a *synthetic replica*. All synchronization of the contents of this backup or synthetic replica volume must have been completed before the original volume can be restored from it. The original volume is immediately available for use while its contents are being restored.

Note: You can perform either a destructive or non-destructive restoration of an original volume from an instant snapshot. Only non-destructive restoration is possible from a space-optimized snapshot. In this case, the snapshot remains in existence after the restoration is complete.

Creating instant snapshots

Note: You need a full license to use this feature.

VxVM allows you to make instant snapshots of volumes by using the `vxsnap` command.

Note: The information in this section also applies to RAID-5 volumes that have been converted to a special layered volume layout by the addition of a DCO and DCO volume. See "Using a DCO and DCO volume with a RAID-5 volume" on page 277 for details.

A plex in a full-sized instant snapshot requires as much space as the original volume. If you instead make a space-optimized instant snapshot of a volume, this only requires enough storage to record the original contents of the parent volume as they are changed during the life of the snapshot.

The recommended approach to performing volume backup from the command line, or from a script, is to use the `vxsnap` command. The `vxsnap prepare` and `make` tasks allow you to back up volumes online with minimal disruption to users.

The `vxsnap prepare` step creates a DCO and DCO volume and associates this with the volume. It also enables Persistent FastResync on the volume.

The `vxsnap make` step creates an instant snapshot that is immediately available for making a backup. After the snapshot has been taken, read requests for data in the original volume are satisfied by reading either from a non-updated region of the original volume, or from the copy of the original contents of an updated region that have been recorded by the snapshot.

Note: Synchronization of a full-sized instant snapshot from the original volume is enabled by default. If you specify the `syncing=no` attribute to `vxsnap make`, this disables synchronization, and the contents of the instant snapshot are unlikely ever to become fully synchronized with the contents of the original volume at the point in time that the snapshot was taken. If you wish to move an instant snapshot volume to another disk group for export to another machine for off-host processing, or to turn it into an independent volume, you must ensure that the snapshot volume has been completely synchronized.

You can immediately retake a full-sized or space-optimized instant snapshot at any time by using the `vxsnap refresh` command. If a fully synchronized instant snapshot is required, you must wait for the new resynchronization to complete.

You can create instant snapshots of volume sets by replacing volume names with volume set names in the `vxsnap` command. For more information, see "Creating instant snapshots of volume sets" on page 334.

Note: When using the `vxsnap prepare` or `vxassist make` commands to make a volume ready for instant snapshot operations, if the specified region size exceeds half the value of the tunable `voliomem_maxpool_sz` (see "voliomem_maxpool_sz" on page 483), the operation succeeds but gives a warning such as the following (for a system where `voliomem_maxpool_sz` is set to 12MB):

```
VxVM vxassist WARNING V-5-1-0 Specified regionsize is
larger than the limit on the system
(voliomem_maxpool_sz/2=6144k).
```

If this message is displayed, `vxsnap make`, `refresh` and `restore` operations on such volumes fail as they might potentially hang the system. Such volumes can be used only for break-off snapshot operations using the `reattach` and `make` operations.

To make the volumes usable for instant snapshot operations, use `vxsnap unprepare` on the volume, and then use `vxsnap prepare` to re-prepare the volume with a region size that is less than half the size of `voliomem_maxpool_sz` (in this example, 1MB):

```
# vxsnap -g mydg -f unprepare vol1
# vxsnap -g mydg prepare vol1 regionsize=1M
```

To create and manage a snapshot of a volume with the `vxsnap` command, follow the procedure in "Preparing to create instant and break-off snapshots" on page 321, and then use one of the procedures described in the following sections:

- "Creating and managing space-optimized instant snapshots" on page 324
- "Creating and managing full-sized instant snapshots" on page 327
- "Creating and managing third-mirror break-off snapshots" on page 329
- "Creating and managing linked break-off snapshot volumes" on page 331

Preparing to create instant and break-off snapshots

To prepare a volume for the creation of instant and break-off snapshots

1 Use the following commands to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume, and to check that FastResync is enabled on the volume:

```
# vxprint -g volumedg -F%instant volume
# vxprint -g volumedg -F%fastresync volume
```

If both commands return a value of `on`, the volume can be used for instant snapshot operations, and you should skip to step 3. Otherwise continue with step 2.

2 To prepare a volume for instant snapshots, use the following command:

```
# vxsnap [-g diskgroup] prepare volume [regionsize=size] \
  [ndcomirs=number] [alloc=storage_attributes]
```

Note: It is only necessary to run the `vxsnap prepare` command on a volume if it does not already have a version 20 DCO volume (for example, if you have run the `vxsnap unprepare` command on the volume). See "Creating a volume with a version 20 DCO volume" on page 252, "Preparing a volume for DRL and instant snapshots" on page 275 and "Removing support for DRL and instant snapshots from a volume" on page 279 for more information.

For example, to prepare the volume, my`vol`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg prepare myvol regionsize=128k ndcomirs=2 \
  alloc=mydg10,mydg11
```

This example creates a DCO object and redundant DCO volume with two plexes located on disks `mydg10` and `mydg11`, and associates them with `myvol`. The region size is also increased to 128KB from the default size of 64KB. The region size must be a power of 2, and be greater than or equal to 16KB. A smaller value requires more disk space for the change maps, but the finer granularity provides faster resynchronization.

3 If you need to create several space-optimized instant snapshots for the volumes in a disk group, you may find it more convenient to create a single shared cache object in the disk group rather than a separate cache object for each snapshot. Follow the procedure in "Creating a shared cache object" on page 322 to create the cache object.

For full-sized instant snapshots and linked break-off snapshots, you must prepare a volume that is to be used as the snapshot volume. This volume must be the same size as the data volume for which the snapshot is being

created, and it must also have the same region size. See “Creating a volume for use as a full-sized instant or linked break-off snapshot” on page 323 for details.

Creating a shared cache object

To create a shared cache object

1 Decide on the following characteristics that you want to allocate to the cache volume that underlies the cache object:

- The size of the cache volume should be sufficient to record changes to the parent volumes during the interval between snapshot refreshes. A suggested value is 10% of the total size of the parent volumes for a refresh interval of 24 hours.
- If redundancy is a desired characteristic of the cache volume, it should be mirrored. This increases the space that is required for the cache volume in proportion to the number of mirrors that it has.
- If the cache volume is mirrored, space is required on at least as many disks as it has mirrors. These disks should not be shared with the disks used for the parent volumes. The disks should also be chosen to avoid impacting I/O performance for critical volumes, or hindering disk group split and join operations.

2 Having decided on its characteristics, use the `vxassist` command to create the volume that is to be used for the cache volume. The following example creates a mirrored cache volume, `cachevol`, with size 1GB in the disk group, `mydg`, on the disks `mydg16` and `mydg17`:

```
# vxassist -g mydg make cachevol 1g layout=mirror \
  init=active mydg16 mydg17
```

The attribute `init=active` is specified to make the cache volume immediately available for use.

3 Use the `vxmake cache` command to create a cache object on top of the cache volume that you created in the previous step:

```
# vxmake [-g diskgroup] cache cache_object \
  cachevolname=volume [regionsize=size] [autogrow=on] \
  [highwatermark=hwmk] [autogrowby=agbvalue] \
  [maxautogrow=maxagbvalue]]
```

If the region size, `regionsize`, is specified, it must be a power of 2, and be greater than or equal to 16KB (`16k`). If not specified, the region size of the cache is set to 64KB.

Note: All space-optimized snapshots that share the cache must have a region size that is equal to or an integer multiple of the region size set on the cache. Snapshot creation also fails if the original volume's region size is smaller than the cache's region size.

If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

If the cache is to be allowed to grow in size as required, specify `autogrow=on`. By default, the ability to automatically grow the cache is turned off.
In the following example, the cache object, `cobjmydg`, is created over the cache volume, `cachevol`, the region size of the cache is set to 32KB, and the `autogrow` feature is enabled:

```
# vxmake -g mydg cache cobjmydg cachevolname=cachevol \
  regionsize=32k autogrow=on
```

4 Having created the cache object, use the following command to enable it:

```
# vxcache [-g diskgroup] start cache_object
```

For example to start the cache object, `cobjmydg`:

```
# vxcache -g mydg start cobjmydg
```

For details of how to remove a cache, see "Removing a cache" on page 347.

Creating a volume for use as a full-sized instant or linked break-off snapshot

To create an empty volume for use by a full-sized instant snapshot or a linked break-off snapshot

1 Use the `vxprint` command on the original volume to find the required size for the snapshot volume.

```
# LEN=`vxprint [-g diskgroup] -F%len volume`
```

Note: The command shown in this and subsequent steps assumes that you are using a Bourne-type shell such as `sh`, `ksh` or `bash`. You may need to modify the command for other shells such as `csh` or `tcsh`.

2 Use the `vxprint` command on the original volume to discover the name of its DCO:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name volume`
```

3 Use the `vxprint` command on the DCO to discover its region size (in blocks):

```
# RSZ=`vxprint [-g diskgroup] -F%regionsz $DCONAME`
```

4 Use the `vxassist` command to create a volume, *snapvol*, of the required size and redundancy, together with a version 20 DCO volume with the correct region size:

```
# vxassist [-g diskgroup] make snapvol $LEN \
  [layout=mirror nmirror=number] logtype=dco drl=off \
  dcoversion=20 [ndcomirror=number] regionsz=$RSZ \
  init=active [storage_attributes]
```

Specify the same number of DCO mirrors (`ndcomirror`) as the number of mirrors in the volume (`nmirror`). The `init=active` attribute is used to make the volume available immediately. You can use storage attributes to specify which disks should be used for the volume.

As an alternative to creating the snapshot volume and its DCO volume in a single step, you can first create the volume, and then prepare it for instant snapshot operations as shown here:

```
# vxassist [-g diskgroup] make snapvol $LEN \
  [layout=mirror nmirror=number] init=active \
  [storage_attributes]
# vxsnap [-g diskgroup] prepare snapvol [ndcomirs=number] \
  regionsize=$RSZ [storage_attributes]
```

Creating and managing space-optimized instant snapshots

Note: Space-optimized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade the performance of the volume.

If you intend to split the volume and snapshot into separate disk groups (for example, to perform off-host processing), you must use a fully synchronized full-sized instant, third-mirror break-off or linked break-off snapshot (which do not require a cache object). You cannot use a space-optimized instant snapshot for this purpose.

Creation of space-optimized snapshots that use a shared cache fails if the region size specified for the volume is smaller than the region size set on the cache.

If the region size of a space-optimized snapshot differs from the region size of the cache, this can degrade the system's performance compared to the case where the region sizes are the same.

For space-optimized instant snapshots that share a cache object, the specified region size must be greater than or equal to the region size specified for the cache object. See "Creating a shared cache object" on page 322 for details.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a space-optimized instant snapshot

1 Use the `vxsnap make` command to create a space-optimized instant snapshot. This snapshot can be created by using an existing cache object in the disk group, or a new cache object can be created for its use.

- To create a space-optimized instant snapshot, *`snapvol`*, that uses a named shared cache object:

  ```
  # vxsnap [-g diskgroup] make source=vol/newvol=snapvol\
  /[cache=cacheobject] [alloc=storage_attributes]
  ```

 For example, to create the space-optimized instant snapshot, `snap3myvol`, of the volume, `myvol`, in the disk group, `mydg`, on the disk `mydg14`, and which uses the shared cache object, `cobjmydg`, use the following command:

  ```
  # vxsnap -g mydg make source=myvol/newvol=snap3myvol\
  /cache=cobjmydg alloc=mydg14
  ```

 For details of how to create a shared cache object, see "Creating a shared cache object" on page 322.

- To create a space-optimized instant snapshot, *`snapvol`*, and also create a cache object for it to use:

  ```
  # vxsnap [-g diskgroup] make source=vol/newvol=snapvol\
  [/cachesize=size][/autogrow=yes][/ncachemirror=number]\
  [alloc=storage_attributes]
  ```

 The `cachesize` attribute determines the size of the cache relative to the size of the volume. The `autogrow` attribute determines whether VxVM will automatically enlarge the cache if it is in danger of overflowing. By default, the cache is not grown.

 Note: If `autogrow` is enabled, but the cache cannot be grown, VxVM disables the oldest and largest snapshot that is using the same cache, and releases its cache space for use.

 The `ncachemirror` attribute specifies the number of mirrors to create in the cache volume. For backup purposes, the default value of 1 should be sufficient.

For example, to create the space-optimized instant snapshot, `snap4myvol`, of the volume, `myvol`, in the disk group, `mydg`, on the disk `mydg15`, and which uses a newly allocated cache object that is 1GB in size, but which can automatically grow in size, use the following command:

```
# vxsnap -g mydg make source=myvol/new=snap4myvol\
/cachesize=1g/autogrow=yes alloc=mydg15
```

Note: If a cache is created implicitly by specifying `cachesize`, and `ncachemirror` is specified to be greater than 1, a DCO is attached to the cache volume to enable dirty region logging (DRL). DRL allows fast recovery of the cache backing store after a system crash. The DCO is allocated on the same disks as those that are occupied by the DCO of the source volume. This is done to allow the cache and the source volume to remain in the same disk group for disk group move, split and join operations.

2 Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command with a VxFS file system:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

3 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

4 You now have the following choices of what to do with a space-optimized instant snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized. See "Refreshing an instant snapshot" on page 337 for details.
- Restore the contents of the original volume from the snapshot volume. The space-optimized instant snapshot remains intact at the end of the operation. See "Restoring a volume from an instant snapshot" on page 340 for details.

Creating and managing full-sized instant snapshots

Note: Full-sized instant snapshots are not suitable for write-intensive volumes (such as for database redo logs) because the copy-on-write mechanism may degrade the performance of the volume.

For full-sized instant snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size. See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 323 for details.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a full-sized instant snapshot

1 To create a full-sized instant snapshot, use the following form of the `vxsnap make` command:

```
# vxsnap [-g diskgroup] make source=volume/snapvol=snapvol\
[/snapdg=snapdiskgroup] [/syncing=off]
```

The command specifies the volume, *snapvol*, that you prepared earlier.

For example, to use the prepared volume, `snap1myvol`, as the snapshot for the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap1myvol
```

For full-sized instant snapshots that are created from an empty volume, background synchronization is enabled by default (equivalent to specifying the `syncing=on` attribute). If you want to move a snapshot into a separate disk group, or to turn it into an independent volume, you must wait for its contents to be synchronized with those of its parent volume.

You can use the `vxsnap syncwait` command to wait for the synchronization of the snapshot volume to be completed, as shown here:

```
# vxsnap [-g diskgroup] syncwait snapvol
```

For example, you would use the following command to wait for synchronization to finish on the snapshot volume, snap2myvol:

```
# vxsnap -g mydg syncwait snap2myvol
```

This command exits (with a return code of zero) when synchronization of the snapshot volume is complete. The snapshot volume may then be moved to another disk group or turned into an independent volume.

If required, you can use the following command to test if the synchronization of a volume is complete:

```
# vxprint [-g diskgroup] -F%incomplete snapvol
```

This command returns the value `off` if synchronization of the volume, *snapvol*, is complete; otherwise, it returns the value `on`.

You can also use the `vxsnap print` command to check on the progress of synchronization as described in "Displaying instant snapshot information" on page 342.

See "Controlling instant snapshot synchronization" on page 344 for more information.

If you do not want to move the snapshot into a separate disk group, or to turn it into an independent volume, specify the `syncing=off` attribute. This avoids creating unnecessary system overhead. For example, to turn off synchronization when creating the snapshot of the volume, `myvol`, you would use the following form of the `vxsnap make` command:

```
# vxsnap -g mydg make source=myvol/snapvol=snap1myvol\
/syncing=off
```

2 Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command with a VxFS file system:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

3 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

4 You now have the following choices of what to do with a full-sized instant snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized. See "Refreshing an instant snapshot" on page 337 for details.
- Reattach some or all of the plexes of the snapshot volume with the original volume. See "Reattaching an instant snapshot" on page 338 for details.
- Restore the contents of the original volume from the snapshot volume. You can choose whether none, a subset, or all of the plexes of the snapshot volume are returned to the original volume as a result of the operation. See "Restoring a volume from an instant snapshot" on page 340 for details.

- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume. See "Dissociating an instant snapshot" on page 340 for details.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes. See "Splitting an instant snapshot hierarchy" on page 341 for details.

Creating and managing third-mirror break-off snapshots

Note: Break-off snapshots are suitable for write-intensive volumes, such as database redo logs.

To turn one or more existing plexes in a volume into a break-off instant snapshot volume, the volume must be a non-layered volume with a `mirror` or `mirror-stripe` layout, or a RAID-5 volume that you have converted to a special layered volume (see "Using a DCO and DCO volume with a RAID-5 volume" on page 277) and then mirrored. The plexes in a volume with a `stripe-mirror` layout are mirrored at the subvolume level, and cannot be broken off.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a third-mirror break-off snapshot

1 To create the snapshot, you can either take some of the existing `ACTIVE` plexes in the volume, or you can use the following command to add new snapshot mirrors to the volume:

```
# vxsnap [-b] [-g diskgroup] addmir volume [nmirror=N] \
  [alloc=storage_attributes]
```

By default, the `vxsnap addmir` command adds one snapshot mirror to a volume unless you use the `nmirror` attribute to specify a different number of mirrors. The mirrors remain in the `SNAPATT` state until they are fully synchronized. The `-b` option can be used to perform the synchronization in the background. Once synchronized, the mirrors are placed in the `SNAPDONE` state.

For example, the following command adds 2 mirrors to the volume, `vol1`, on disks `mydg10` and `mydg11`:

```
# vxsnap -g mydg addmir vol1 nmirror=2 alloc=mydg10,mydg11
```

If you specify the -b option to the vxsnap addmir command, you can use the vxsnap snapwait command to wait for synchronization of the snapshot plexes to complete, as shown in this example:

```
# vxsnap -g mydg snapwait vol1 nmirror=2
```

2 To create a third-mirror break-off snapshot, use the following form of the vxsnap make command.

```
# vxsnap [-g diskgroup] make source=volume[/newvol=snapvol]\
{/plex=plex1[,plex2,...]|/nmirror=number]}
```

Either of the following attributes may be specified to create the new snapshot volume, *snapvol*, by breaking off one or more existing plexes in the original volume:

plex Specifies the plexes in the existing volume that are to be broken off. This attribute can only be used with plexes that are in the ACTIVE state.

nmirror Specifies how many plexes are to be broken off. This attribute can only be used with plexes that are in the SNAPDONE state. (Such plexes could have been added to the volume by using the vxsnap addmir command.)

Snapshots that are created from one or more ACTIVE or SNAPDONE plexes in the volume are already synchronized by definition.

For backup purposes, a snapshot volume with one plex should be sufficient.

For example, to create the instant snapshot volume, snap2myvol, of the volume, myvol, in the disk group, mydg, from a single existing plex in the volume, use the following command:

```
# vxsnap -g mydg make source=myvol/newvol=snap2myvol\
/nmirror=1
```

The next example shows how to create a mirrored snapshot from two existing plexes in the volume:

```
# vxsnap -g mydg make source=myvol/newvol=snap2myvol\
/plex=myvol-03,myvol-04
```

3 Use fsck (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command with a VxFS file system:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

4 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

5 You now have the following choices of what to do with a third-mirror break-off snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If

synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized. See “Refreshing an instant snapshot” on page 337 for details.

- Reattach some or all of the plexes of the snapshot volume with the original volume. See “Reattaching an instant snapshot” on page 338 for details.
- Restore the contents of the original volume from the snapshot volume. You can choose whether none, a subset, or all of the plexes of the snapshot volume are returned to the original volume as a result of the operation. See “Restoring a volume from an instant snapshot” on page 340 for details.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume. See “Dissociating an instant snapshot” on page 340 for details.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes. See “Splitting an instant snapshot hierarchy” on page 341 for details.

Creating and managing linked break-off snapshot volumes

Note: Break-off snapshots are suitable for write-intensive volumes, such as database redo logs.

For linked break-off snapshots, you must prepare a volume that is to be used as the snapshot volume. This must be the same size as the volume for which the snapshot is being created, and it must also have the same region size. See “Creating a volume for use as a full-sized instant or linked break-off snapshot” on page 323 for details.

The attributes for a snapshot are specified as a tuple to the `vxsnap make` command. This command accepts multiple tuples. One tuple is required for each snapshot that is being created. Each element of a tuple is separated from the next by a slash character (/). Tuples are separated by white space.

To create and manage a linked break-off snapshot

1 Use the following command to link the prepared snapshot volume, *snapvol*, to the data volume:

```
# vxsnap [-g diskgroup] [-b] addmir volume mirvol=snapvol \
```

[mirdg=*snapdg*]

The optional `mirdg` attribute can be used to specify the snapshot volume's current disk group, *snapdg*. The `-b` option can be used to perform the synchronization in the background. If the `-b` option is not specified, the command does not return until the link becomes `ACTIVE`.

For example, the following command links the prepared volume, `prepsnap`, in the disk group, `mysnapdg`, to the volume, `vol1`, in the disk group, `mydg`:

```
# vxsnap -g mydg -b addmir vol1 mirvol=prepsnap \
  mirdg=mysnapdg
```

If the `-b` option is specified, you can use the `vxsnap snapwait` command to wait for the synchronization of the linked snapshot volume to complete, as shown in this example:

```
# vxsnap -g mydg snapwait vol1 mirvol=prepsnap \
  mirdg=mysnapvoldg
```

2 To create a linked break-off snapshot, use the following form of the `vxsnap make` command.

```
# vxsnap [-g diskgroup] make source=volume/snapvol=snapvol\
[/snapdg=snapdiskgroup]
```

The `snapdg` attribute must be used to specify the snapshot volume's disk group if this is different from that of the data volume.

For example, to use the prepared volume, `prepsnap`, as the snapshot for the volume, `vol1`, in the disk group, `mydg`, use the following command:

```
# vxsnap -g mydg make \
  source=vol1/snapvol=prepsnap/snapdg=mysnapdg
```

3 Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command with a VxFS file system:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

4 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

5 You now have the following choices of what to do with a linked break-off snapshot:

- Refresh the contents of the snapshot. This creates a new point-in-time image of the original volume ready for another backup. If synchronization was already in progress on the snapshot, this operation may result in large portions of the snapshot having to be resynchronized. See "Refreshing an instant snapshot" on page 337 for details.

Note: This operation is not possible if the linked volume and snapshot are in different disk groups.

- Reattach the snapshot volume with the original volume. See "Reattaching a linked break-off snapshot volume" on page 339 for details.
- Dissociate the snapshot volume entirely from the original volume. This may be useful if you want to use the copy for other purposes such as testing or report generation. If desired, you can delete the dissociated volume. See "Dissociating an instant snapshot" on page 340 for details.
- If the snapshot is part of a snapshot hierarchy, you can also choose to split this hierarchy from its parent volumes. See "Splitting an instant snapshot hierarchy" on page 341 for details.

Creating multiple instant snapshots

To make it easier to create snapshots of several volumes at the same time, the `vxsnap make` command accepts multiple tuples that define the source and snapshot volumes names as their arguments. For example, to create three instant snapshots, each with the same redundancy, from specified storage, the following form of the command can be used:

```
# vxsnap [-g diskgroup] make source=vol1/snapvol=snapvol1\
  source=vol2/snapvol=snapvol2 source=vol3/snapvol=snapvol3
```

The snapshot volumes (*snapvol1*, *snapvol2* and so on) must have been prepared in advance as described in "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 323. The specified source volumes (*vol1*, *vol2* and so on) may be the same volume or they can be different volumes.

If all the snapshots are to be space-optimized and to share the same cache, the following form of the command can be used:

```
# vxsnap [-g diskgroup] make \
  source=vol1/newvol=snapvol1/cache=cacheobj \
  source=vol2/newvol=snapvol2/cache=cacheobj \
  source=vol3/newvol=snapvol3/cache=cacheobj \
  [alloc=storage_attributes]
```

The `vxsnap make` command also allows the snapshots to be of different types, have different redundancy, and be configured from different storage, as shown here:

```
# vxsnap [-g diskgroup] make source=vol1/snapvol=snapvol1 \
  source=vol2[/newvol=snapvol2]/cache=cacheobj\
[/alloc=storage_attributes2][/nmirror=number2]
  source=vol3[/newvol=snapvol3][/alloc=storage_attributes3]\
/nmirror=number3
```

In this example, *snapvol1* is a full-sized snapshot that uses a prepared volume, *snapvol2* is a space-optimized snapshot that uses a prepared cache, and *snapvol3* is a break-off full-sized snapshot that is formed from plexes of the original volume.

An example of where you might want to create mixed types of snapshots at the same time is when taking snapshots of volumes containing database redo logs and database tables:

```
# vxsnap -g mydg make \
  source=logv1/newvol=snplogv1/drl=sequential/nmirror=1 \
  source=logv2/newvol=snplogv2/drl=sequential/nmirror=1 \
  source=datav1/newvol=snpdatav1/cache=mydgcobj/drl=on \
  source=datav2/newvol=snpdatav2/cache=mydgcobj/drl=on
```

In this example, sequential DRL is enabled for the snapshots of the redo log volumes, and normal DRL is applied to the snapshots of the volumes that contain the database tables. The two space-optimized snapshots are configured to share the same cache object in the disk group. Also note that break-off snapshots are used for the redo logs as such volumes are write intensive.

Creating instant snapshots of volume sets

Volume set names can be used in place of volume names with the following `vxsnap` operations on instant snapshots: `addmir`, `dis`, `make`, `prepare`, `reattach`, `refresh`, `restore`, `rmmir`, `split`, `syncpause`, `syncresume`, `syncstart`, `syncstop`, `syncwait`, and `unprepare`.

The procedure for creating an instant snapshot of a volume set is the same as that for a standalone volume. However, there are certain restrictions if a full-sized instant snapshot is to be created from a prepared volume set. A full-sized instant snapshot of a volume set must itself be a volume set with the same number of volumes, and the same volume sizes and index numbers as the parent. For example, if a volume set contains three volumes with sizes 1GB, 2GB and 3GB, and indexes 0, 1 and 2 respectively, then the snapshot volume set must have three volumes with the same sizes matched to the same set of index numbers. The corresponding volumes in the parent and snapshot volume sets are also subject to the same restrictions as apply between standalone volumes and their snapshots.

You can use the `vxvset list` command to verify that the volume sets have identical characteristics as shown in this example:

```
# vxvset -g mydg list vset1
VOLUME        INDEX       LENGTH      KSTATE      CONTEXT
vol_0         0           204800      ENABLED     -
vol_1         1           409600      ENABLED     -
vol_2         2           614400      ENABLED     -

# vxvset -g mydg list snapvset1
```

```
VOLUME        INDEX       LENGTH      KSTATE      CONTEXT
svol_0        0           204800      ENABLED     -
svol_1        1           409600      ENABLED     -
svol_2        2           614400      ENABLED     -
```

A full-sized instant snapshot of a volume set can be created using a prepared volume set in which each volume is the same size as the corresponding volume in the parent volume set. Alternatively, you can use the `nmirrors` attribute to specify the number of plexes that are to be broken off provided that sufficient plexes exist for each volume in the volume set.

The following example shows how to prepare a source volume set, `vset1`, and an identical volume set, `snapvset1`, which is then used to create the snapshot:

```
# vxsnap -g mydg prepare vset1
# vxsnap -g mydg prepare snapvset1
# vxsnap -g mydg make source=vset1/snapvol=snapvset1
```

To create a full-sized third-mirror break-off snapshot, you must ensure that each volume in the source volume set contains sufficient plexes. The following example shows how to achieve this by using the `vxsnap` command to add the required number of plexes before breaking off the snapshot:

```
# vxsnap -g mydg prepare vset2
# vxsnap -g mydg addmir vset2 nmirror=1
# vxsnap -g mydg make source=vset2/newvol=snapvset2/nmirror=1
```

See "Adding snapshot mirrors to a volume" on page 336 for more information about adding plexes to volumes or to volume sets.

To create a space-optimized instant snapshot of a volume set, the commands are again identical to those for a standalone volume as shown in these examples:

```
# vxsnap -g mydg prepare vset3
# vxsnap -g mydg make source=vset3/newvol=snapvset3/
  cachesize=20m

# vxsnap -g mydg prepare vset4
# vxsnap -g mydg make source=vset4/newvol=snapvset4/cache=mycobj
```

Here a new cache object is created for the volume set, `vset3`, and an existing cache object, `mycobj`, is used for `vset4`.

See "Creating and administering volume sets" on page 361 for more information on creating and administering volume sets.

Adding snapshot mirrors to a volume

If you are going to create a full-sized break-off snapshot volume, you can use the following command to add new snapshot mirrors to a volume:

```
# vxsnap [-b] [-g diskgroup] addmir volume|volume_set \
  [nmirror=N] [alloc=storage_attributes]
```

Note: The volume must have been prepared using the `vxsnap prepare` command as described in "Preparing a volume for DRL and instant snapshots" on page 275.

If a volume set name is specified instead of a volume, the specified number of plexes is added to each volume in the volume set.

By default, the `vxsnap addmir` command adds one snapshot mirror to a volume unless you use the `nmirror` attribute to specify a different number of mirrors. The mirrors remain in the `SNAPATT` state until they are fully synchronized. The `-b` option can be used to perform the synchronization in the background. Once synchronized, the mirrors are placed in the `SNAPDONE` state.

For example, the following command adds 2 mirrors to the volume, `vol1`, on disks `mydg10` and `mydg11`:

```
# vxsnap -g mydg addmir vol1 nmirror=2 alloc=mydg10,mydg11
```

Note: This command is similar in usage to the `vxassist snapstart` command, and supports the traditional third-mirror break-off snapshot model. As such, it does not provide an instant snapshot capability.

Once you have added one or more snapshot mirrors to a volume, you can use the `vxsnap make` command with either the `nmirror` attribute or the `plex` attribute to create the snapshot volumes.

Removing a snapshot mirror

To remove a single snapshot mirror from a volume, use this command:

```
# vxsnap [-g diskgroup] rmmir volume|volume_set
```

For example, the following command removes a snapshot mirror from the volume, `vol1`:

```
# vxsnap -g mydg rmmir vol1
```

Note: This command is similar in usage to the `vxassist snapabort` command.

If a volume set name is specified instead of a volume, a mirror is removed from each volume in the volume set.

Removing a linked break-off snapshot volume

To remove a linked break-off snapshot volume from a volume, use this command:

```
# vxsnap [-g diskgroup] rmmir volume|volume_set mirvol=snapvol \
  [mirdg=snapdiskgroup]
```

The `mirvol` and optional `mirdg` attributes specify the snapshot volume, *snapvol*, and its disk group, *snapdiskgroup*. For example, the following command removes a linked snapshot volume, `prepsnap`, from the volume, `vol1`:

```
# vxsnap -g mydg rmmir vol1 mirvol=prepsnap mirdg=mysnapdg
```

Adding a snapshot to a cascaded snapshot hierarchy

To create a snapshot and push it onto a snapshot hierarchy between the original volume and an existing snapshot volume, specify the name of the existing snapshot volume as the value of the `infrontof` attribute to the `vxsnap make` command. The following example shows how to place the space-optimized snapshot, `thurs_bu`, of the volume, `dbvol`, in front of the earlier snapshot, `wed_bu`:

```
# vxsnap -g dbdg make source=dbvol/newvol=thurs_bu/\
  infrontof=wed_bu/cache=dbdgcache
```

Similarly, the next snapshot that is taken, `fri_bu`, is placed in front of `thurs_bu`:

```
# vxsnap -g dbdg make source=dbvol/newvol=fri_bu/\
  infrontof=thurs_bu/cache=dbdgcache
```

For more information on the application of cascaded snapshots, see "Cascaded snapshots" on page 312.

Refreshing an instant snapshot

Refreshing an instant snapshot replaces it with another point-in-time copy of a parent volume. To refresh one or more snapshots and make them immediately available for use, use the following command:

```
# vxsnap [-g diskgroup] refresh snapvolume|snapvolume_set \
  source=volume|volume_set [[snapvol2 source=vol2]...] \
  [syncing=yes|no]
```

If the source volume is not specified, the immediate parent of the snapshot is used. For full-sized instant snapshots, resynchronization is started by default.

To disable resynchronization, specify the `syncing=no` attribute. This attribute is not supported for space-optimized snapshots.

Note: The snapshot being refreshed must not be open to any application. For example, any file system configured on the volume must first be unmounted.

It is possible to refresh a volume from an unrelated volume provided that their sizes are compatible.

You can use the `vxsnap syncwait` command to wait for the synchronization of the snapshot volume to be completed, as shown here:

```
# vxsnap [-g diskgroup] syncwait snapvol
```

See "Controlling instant snapshot synchronization" on page 344 for more information.

Reattaching an instant snapshot

Note: This operation is not supported for space-optimized instant snapshots.

Using the following command, some or all plexes of an instant snapshot may be reattached to the specified original volume, or to a source volume in the snapshot hierarchy above the snapshot volume:

```
# vxsnap [-g diskgroup] reattach snapvolume|snapvolume_set \
  source=volume|volume_set [nmirror=number]
```

By default, all the plexes are reattached, which results in the removal of the snapshot. If required, the number of plexes to be reattached may be specified as the value assigned to the `nmirror` attribute.

Note: The snapshot being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

It is possible to reattach a volume to an unrelated volume provided that their volume sizes and region sizes are compatible.

For example the following command reattaches one plex from the snapshot volume, `snapmyvol`, to the volume, `myvol`:

```
# vxsnap -g mydg reattach snapmyvol source=myvol nmirror=1
```

While the reattached plexes are being resynchronized from the data in the parent volume, they remain in the `SNAPTMP` state. After resynchronization is complete, the plexes are placed in the `SNAPDONE` state. You can use the `vxsnap`

snapwait command (but not vxsnap syncwait) to wait for the resynchronization of the reattached plexes to complete, as shown here:

```
# vxsnap -g mydg snapwait myvol nmirror=1
```

Note: If the volume and its snapshot have both been resized (to an identical smaller or larger size) before performing the reattachment, a fast resynchronization can still be performed. A full resynchronization is not required. Version 20 DCO volumes are resized proportionately when the associated data volume is resized. For version 0 DCO volumes, the FastResync maps stay the same size, but the region size is recalculated, and the locations of the dirty bits in the existing maps are adjusted. In both cases, new regions are marked as dirty in the maps.

Reattaching a linked break-off snapshot volume

Unlike other types of snapshot, the reattachment operation for linked break-off snapshot volumes does not return the plexes of the snapshot volume to the parent volume. The link relationship is re-established that makes the snapshot volume a mirror of the parent volume, and this allows the snapshot data to be resynchronized. However, the snapshot volume is only readopted by its parent volume if they are both in the same disk group.

To reattach a linked break-off snapshot volume, use the following form of the vxsnap reattach command:

```
# vxsnap [-g snapdiskgroup] reattach snapvolume|snapvolume_set \
  source=volume|volume_set [sourcedg=diskgroup]
```

The sourcedg attribute must be used to specify the data volume's disk group if this is different from the snapshot volume's disk group, *snapdiskgroup*.

Note: The snapshot being reattached must not be open to any application. For example, any file system configured on the snapshot volume must first be unmounted.

It is possible to reattach a volume to an unrelated volume provided that their sizes and region sizes are compatible.

For example the following command reattaches the snapshot volume, prepsnap, in the disk group, snapdg, to the volume, myvol, in the disk group, mydg:

```
# vxsnap -g snapdg reattach prepsnap source=myvol sourcedg=mydg
```

After resynchronization of the snapshot volume is complete, the link is placed in the ACTIVE state. You can use the vxsnap snapwait command (but not vxsnap

syncwait) to wait for the resynchronization of the reattached volume to complete, as shown here:

```
# vxsnap -g snapdg snapwait myvol mirvol=prepsnap
```

Restoring a volume from an instant snapshot

It may sometimes be desirable to reinstate the contents of a volume from a backup or modified replica in a snapshot volume. The following command may be used to restore one or more volumes from the specified snapshots:

```
# vxsnap [-g diskgroup] restore volume|volume_set \
 source=snapvolume|snapvolume_set \
 [[volume2|volume_set2 source=snapvolume2|snapvolume_set2]...]\
 [destroy=yes|no] [syncing=yes|no] [nmirror=number]
```

For a full-sized instant snapshot, some or all of its plexes may be reattached to the parent volume or to a specified source volume in the snapshot hierarchy above the snapshot volume. If destroy=yes is specified, all the plexes of the full-sized instant snapshot are reattached and the snapshot volume is removed.

For a space-optimized instant snapshot, the cached data is used to recreate the contents of the specified volume. The space-optimized instant snapshot remains unchanged by the restore operation.

Note: For this operation to succeed, the volume that is being restored and the snapshot volume must not be open to any application. For example, any file systems that are configured on either volume must first be unmounted.

It is not possible to restore a volume from an unrelated volume.

The destroy and nmirror attributes are not supported for space-optimized instant snapshots.

The following example demonstrates how to restore the volume, myvol, from the space-optimized snapshot, snap3myvol.

```
# vxsnap -g mydg restore myvol source=snap3myvol
```

Dissociating an instant snapshot

The following command breaks the association between a full-sized instant snapshot volume, *snapvol*, and its parent volume, so that the snapshot may be used as an independent volume:

```
# vxsnap [-f] [-g diskgroup] dis snapvolume|snapvolume_set
```

This operation fails if the snapshot, *snapvol*, has a snapshot hierarchy below it that contains unsynchronized snapshots. If this happens, the dependent snapshots must be fully synchronized from *snapvol*. When no dependent

snapshots remain, *snapvol* may be dissociated. The snapshot hierarchy is then adopted by *snapvol*'s parent volume.

Note: To be usable after dissociation, the snapshot volume and any snapshots in the hierarchy must have been fully synchronized. See "Controlling instant snapshot synchronization" on page 344 for more information. In addition, you cannot dissociate a snapshot if synchronization of any of the dependent snapshots in the hierarchy is incomplete. If an incomplete snapshot is dissociated, it is unusable and should be deleted as described in "Removing an instant snapshot" on page 341.

The following command dissociates the snapshot, `snap2myvol`, from its parent volume:

```
# vxsnap -g mydg dis snap2myvol
```

Note: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the `-f` (force) option must be specified.

Removing an instant snapshot

When you have dissociated a full-sized instant snapshot, you can use the `vxedit` command to delete it altogether, as shown in this example:

```
# vxedit -g mydg -r rm snap2myvol
```

You can also use this command to remove a space-optimized instant snapshot from its cache. For details of how to remove a cache, see "Removing a cache" on page 347.

Splitting an instant snapshot hierarchy

Note: This operation is not supported for space-optimized instant snapshots.

The following command breaks the association between a snapshot hierarchy that has the snapshot volume, *snapvol*, at its head, and its parent volume, so that the snapshot hierarchy may be used independently of the parent volume:

```
# vxsnap [-f] [-g diskgroup] split snapvolume|snapvolume_set
```

Note: The topmost snapshot volume in the hierarchy must have been fully synchronized for this command to succeed. Snapshots that are lower down in the hierarchy need not have been fully resynchronized. See "Controlling instant snapshot synchronization" on page 344 for more information.

The following command splits the snapshot hierarchy under `snap2myvol` from its parent volume:

```
# vxsnap -g mydg split snap2myvol
```

Note: When applied to a volume set or to a component volume of a volume set, this operation can result in inconsistencies in the snapshot hierarchy in the case of a system crash or hardware failure. If the operation is applied to a volume set, the `-f` (force) option must be specified.

Displaying instant snapshot information

The `vxsnap print` command may be used to display information about the snapshots that are associated with a volume.

```
# vxsnap [-g diskgroup] print [vol]
```

This command shows the percentage progress of the synchronization of a snapshot or volume. If no volume is specified, information about the snapshots for all the volumes in a disk group is displayed. The following example shows a volume, `vol1`, which has a full-sized snapshot, `snapvol1` whose contents have not been synchronized with `vol1`:

```
# vxsnap -g mydg print
NAME      SNAPOBJECT    TYPE    PARENT    SNAPSHOT    %DIRTY    %VALID

vol1      --            volume  --        --          --        100
          snapvol1_snp1 volume  --        snapvol1    1.30      --
snapvol1  vol1_snp1     volume  vol1      --          1.30      1.30
```

The `%DIRTY` value for `snapvol1` shows that its contents have changed by 1.30% when compared with the contents of `vol1`. As `snapvol1` has not been synchronized with `vol1`, the `%VALID` value is the same as the `%DIRTY` value. If the snapshot were partly synchronized, the `%VALID` value would lie between the `%DIRTY` value and 100%. If the snapshot were fully synchronized, the `%VALID` value would be 100%. The snapshot could then be made independent or moved into another disk group.

Additional information about the snapshots of volumes and volume sets can be obtained by using the `-n` option with the `vxsnap print` command:

```
# vxsnap [-g diskgroup] -n [-l] [-v] [-x] print [vol]
```

Alternatively, you can use the `vxsnap list` command, which is an alias for the `vxsnap -n print` command:

```
# vxsnap [-g diskgroup] [-l] [-v] [-x] list [vol]
```

The following output is an example of using this command on the disk group `dg1`:

```
# vxsnap -g dg -vx list

NAME    DG    OBJTYPE SNAPTYPE  PARENT  PARENTDG  SNAPDATE        CHANGE_DATA   SYNCED_DATA
vol     dg1   vol     -         -       -         -               -             10G (100%)
svol1   dg2   vol     fullinst  vol     dg1       2006/2/1 12:29  20M (0.2%)    60M (0.6%)
svol2   dg1   vol     mirbrk    vol     dg1       2006/2/1 12:29  120M (1.2%)   10G (100%)
svol3   dg2   vol     volbrk    vol     dg1       2006/2/1 12:29  105M (1.1%)   10G (100%)
svol21  dg1   vol     spaceopt  svol2   dg1       2006/2/1 12:29  52M (0.5%)    52M (0.5%)
vol-02  dg1   plex    snapmir   vol     dg1       -               -             56M (0.6%)
mvol    dg2   vol     mirvol    vol     dg1       -               -             58M (0.6%)
vset1   dg1   vset    -         -       -         -               -             2G (100%)
v1      dg1   compvol -         -       -         -               -             1G (100%)
v2      dg1   compvol -         -       -         -               -             1G (100%)
svset1  dg1   vset    mirbrk    vset    dg1       2006/2/1 12:29  1G (50%)      2G (100%)
sv1     dg1   compvol mirbrk    v1      dg1       2006/2/1 12:29  512M (50%)    1G (100%)
sv2     dg1   compvol mirbrk    v2      dg1       2006/2/1 12:29  512M (50%)    1G (100%)
vol-03  dg1   plex    detmir    vol     dg1       -               20M (0.2%)    -
mvol2   dg2   vol     detvol    vol     dg1       -               20M (0.2%)    -
```

This shows that the volume `vol` has three full-sized snapshots, `svol1`, `svol2` and `svol3`, which are of types full-sized instant (`fullinst`), mirror break-off (`mirbrk`) and linked break-off (`volbrk`). It also has one snapshot plex (`snapmir`), `vol-02`, and one linked mirror volume (`mirvol`), `mvol`. The snapshot `svol2` itself has a space-optimized instant snapshot (`spaceopt`), `svol21`. There is also a volume set, `vset1`, with component volumes `v1` and `v2`. This volume set has a mirror break-off snapshot, `svset1`, with component volumes `sv1` and `sv2`. The last two entries show a detached plex, `vol-03`, and a detached mirror volume, `mvol2`, which have `vol` as their parent volume. These snapshot objects may have become detached due to an I/O error, or, in the case of the plex, by running the `vxplex det` command.

The `CHANGE_DATA` column shows the approximate difference between the current contents of the snapshot and its parent volume. This corresponds to the amount of data that would have to be resynchronized to make the contents the same again.

The `SYNCED_DATA` column shows the approximate progress of synchronization since the snapshot was taken.

The `-l` option can be used to obtain a longer form of the output listing instead of the tabular form.

The `-x` option expands the output to include the component volumes of volume sets.

See the vxsnap(1M) manual page for more information about using the vxsnap print and vxsnap list commands.

Controlling instant snapshot synchronization

Note: Synchronization of the contents of a snapshot with its original volume is not possible for space-optimized instant snapshots.

The commands in this section cannot be used to control the synchronization of linked break-off snapshots.

By default, synchronization is enabled for the vxsnap reattach, refresh and restore operations on instant snapshots. Otherwise, synchronization is disabled unless you specify the syncing=yes attribute to the vxsnap command.

The following table shows the commands that are provided for controlling the synchronization manually.

Command	Description
vxsnap [-g *diskgroup*] syncpause *vol\|vol_set*	Pause synchronization of a volume.
vxsnap [-g *diskgroup*] syncresume \ *vol\|vol_set*	Resume synchronization of a volume.
vxsnap [-b] [-g *diskgroup*] syncstart \ *vol\|vol_set*	Start synchronization of a volume. The -b option puts the operation in the background.
vxsnap [-g *diskgroup*] syncstop *vol\|vol_set*	Stop synchronization of a volume.
vxsnap [-g *diskgroup*] syncwait *vol\|vol_set*	Exit when synchronization of a volume is complete. An error is returned if *vol* is invalid (for example, it is a space-optimized snapshot), or if *vol* is not being synchronized. **Note:** You cannot use this command to wait for synchronization of reattached plexes to complete.

The vxsnap snapwait command is provided to wait for the link between new linked break-off snapshots to become ACTIVE, or for reattached snapshot plexes to reach the SNAPDONE state following resynchronization. See “Creating and managing linked break-off snapshot volumes” on page 331, “Reattaching an

instant snapshot" on page 338 and "Reattaching a linked break-off snapshot volume" on page 339 for details.

Improving the performance of snapshot synchronization

Two optional arguments to the `-o` option are provided to help optimize the performance of synchronization when using the `make`, `refresh`, `restore` and `syncstart` operations:

`iosize=`*`size`* Specifies the size of each I/O request that is used when synchronizing the regions of a volume. Specifying a larger size causes synchronization to complete sooner, but with greater impact on the performance of other processes that are accessing the volume. The default *size* of 1m (1MB) is suggested as the minimum value for high-performance array and controller hardware. The specified value is rounded to a multiple of the volume's region size.

`slow=`*`iodelay`* Specifies the delay in milliseconds between synchronizing successive sets of regions as specified by the value of `iosize`. This can be used to change the impact of synchronization on system performance. The default value of *iodelay* is 0 milliseconds (no delay). Increasing this value slows down synchronization, and reduces the competition for I/O bandwidth with other processes that may be accessing the volume.

Options may be combined as shown in the following examples:

```
# vxsnap -g mydg -o iosize=2m,slow=100 make \
  source=myvol/snapvol=snap2myvol/syncing=on

# vxsnap -g mydg -o iosize=10m,slow=250 syncstart snap2myvol
```

Note: These optional parameters only affect the synchronization of full-sized instant snapshots. They are not supported for space-optimized snapshots.

Listing the snapshots created on a cache

To list the space-optimized instant snapshots that have been created on a cache object, use the following command:

```
# vxcache [-g diskgroup] listvol cache_object
```

The snapshot names are printed as a space-separated list ordered by timestamp. If two or more snapshots have the same timestamp, these snapshots are sorted in order of decreasing size.

Tuning the autogrow attributes of a cache

The highwatermark, autogrowby and maxautogrow attributes determine how the VxVM cache daemon (vxcached) maintains the cache if the autogrow feature has been enabled and vxcached is running:

- When cache usage reaches the high watermark value, highwatermark (default value is 90 percent), vxcached grows the size of the cache volume by the value of autogrowby (default value is 20% of the size of the cache volume in blocks). The new required cache size cannot exceed the value of maxautogrow (default value is twice the size of the cache volume in blocks).
- When cache usage reaches the high watermark value, and the new required cache size would exceed the value of maxautogrow, vxcached deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted.

If the autogrow feature has been disabled:

- When cache usage reaches the high watermark value, vxcached deletes the oldest snapshot in the cache. If there are several snapshots with the same age, the largest of these is deleted. If there is only a single snapshot, this snapshot is detached and marked as invalid.

Note: The vxcached daemon does not remove snapshots that are currently open, and it does not remove the last or only snapshot in the cache.

If the cache space becomes exhausted, the snapshot is detached and marked as invalid. If this happens, the snapshot is unrecoverable and must be removed. Enabling the autogrow feature on the cache helps to avoid this situation occurring. However, for very small caches (of the order of a few megabytes), it is possible for the cache to become exhausted before the system has time to respond and grow the cache. In such cases, either increase the size of the cache manually as described in "Growing and shrinking a cache" on page 347, or use the vxcache set command to reduce the value of highwatermark as shown in this example:

```
# vxcache -g mydg set highwatermark=60 cobjmydg
```

You can use the maxautogrow attribute to limit the maximum size to which a cache can grow. To estimate this size, consider how much the contents of each source volume are likely to change between snapshot refreshes, and allow some additional space for contingency.

If necessary, you can use the vxcache set command to change other autogrow attribute values for a cache. See the vxcache(1M) manual page for details.

Caution: Ensure that the cache is sufficiently large, and that the `autogrow` attributes are configured correctly for your needs.

Growing and shrinking a cache

You can use the `vxcache` command to increase the size of the cache volume that is associated with a cache object:

```
# vxcache [-g diskgroup] growcacheto cache_object size
```

For example, to increase the size of the cache volume associated with the cache object, `mycache`, to 2GB, you would use the following command:

```
# vxcache -g mydg growcacheto mycache 2g
```

To grow a cache by a specified amount, use the following form of the command shown here:

```
# vxcache [-g diskgroup] growcacheby cache_object size
```

For example, the following command increases the size of `mycache` by 1GB:

```
# vxcache -g mydg growcacheby mycache 1g
```

You can similarly use the `shrinkcacheby` and `shrinkcacheto` operations to reduce the size of a cache. See the `vxcache`(1M) manual page for more information.

Removing a cache

To remove a cache completely, including the cache object, its cache volume and all space-optimized snapshots that use the cache:

1. Run the following command to find out the names of the top-level snapshot volumes that are configured on the cache object:

   ```
   # vxprint -g diskgroup -vne \
     "v_plex.pl_subdisk.sd_dm_name ~ /cache_object/"
   ```

 where *cache_object* is the name of the cache object.

2. Remove all the top-level snapshots and their dependent snapshots (this can be done with a single command):

   ```
   # vxedit -g diskgroup -r rm snapvol ...
   ```

 where *snapvol* is the name of a top-level snapshot volume.

3. Stop the cache object:

   ```
   # vxcache -g diskgroup stop cache_object
   ```

4. Finally, remove the cache object and its cache volume:

   ```
   # vxedit -g diskgroup -r rm cache_object
   ```

Creating traditional third-mirror break-off snapshots

VxVM provides third-mirror break-off snapshot images of volume devices using `vxassist` and other commands.

Note: To enhance the efficiency and usability of volume snapshots, turn on FastResync as described in "Enabling FastResync on a volume" on page 292. If Persistent FastResync is required, you must associate a version 0 DCO with the volume as described in "Adding a version 0 DCO and DCO volume" on page 356.

You need a full license to use this feature.

The procedure described in this section requires a plex that is large enough to store the complete contents of the volume. For details of a method that uses space-optimized snapshots, see "Creating instant snapshots" on page 319.

The recommended approach to performing volume backup from the command line, or from a script, is to use the `vxassist` command. The `vxassist snapstart, snapwait`, and `snapshot` tasks allow you to back up volumes online with minimal disruption to users.

The `vxassist snapshot` procedure consists of two steps:

- Run `vxassist snapstart` to create a snapshot mirror.
- Run `vxassist snapshot` to create a snapshot volume.

The `vxassist snapstart` step creates a write-only backup plex which gets attached to and synchronized with the volume. When synchronized with the volume, the backup plex is ready to be used as a `snapshot` mirror. The end of the update procedure is indicated by the new `snapshot` mirror changing its state to SNAPDONE. This change can be tracked by the `vxassist snapwait` task, which waits until at least one of the mirrors changes its state to SNAPDONE. If the attach process fails, the `snapshot` mirror is removed and its space is released.

Note: If the `snapstart` procedure is interrupted, the snapshot mirror is automatically removed when the volume is started.

Once the `snapshot` mirror is synchronized, it continues being updated until it is detached. You can then select a convenient time at which to create a `snapshot` volume as an image of the existing volume. You can also ask users to refrain from using the system during the brief time required to perform the `snapshot` (typically less than a minute). The amount of time involved in

creating the snapshot mirror is long in contrast to the brief amount of time that it takes to create the snapshot volume.

The online backup procedure is completed by running the vxassist snapshot command on a volume with a SNAPDONE mirror. This task detaches the finished snapshot (which becomes a normal mirror), creates a new normal volume and attaches the snapshot mirror to the snapshot volume. The snapshot then becomes a normal, functioning volume and the state of the snapshot is set to ACTIVE.

To back up a volume using the vxassist command

1 Create a snapshot mirror for a volume using the following command:

```
# vxassist [-b] [-g diskgroup] snapstart [nmirror=N] volume
```

For example, to create a snapshot mirror of a volume called voldef, use the following command:

```
# vxassist [-g diskgroup] snapstart voldef
```

The vxassist snapstart task creates a write-only mirror, which is attached to and synchronized from the volume to be backed up.

Note: By default, VxVM attempts to avoid placing snapshot mirrors on a disk that already holds any plexes of a data volume. However, this may be impossible if insufficient space is available in the disk group. In this case, VxVM uses any available space on other disks in the disk group. If the snapshot plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move a snapshot volume into another disk group as described in "Moving DCO volumes between disk groups" on page 200. To override the default storage allocation policy, you can use storage attributes to specify explicitly which disks to use for the snapshot plexes. See "Creating a volume on specific disks" on page 244 for more information.

If you start vxassist snapstart in the background using the -b option, you can use the vxassist snapwait command to wait for the creation of the mirror to complete as shown here:

```
# vxassist [-g diskgroup] snapwait volume
```

If vxassist snapstart is not run in the background, it does not exit until the mirror has been synchronized with the volume. The mirror is then ready to be used as a plex of a snapshot volume. While attached to the original volume, its contents continue to be updated until you take the snapshot.

Use the nmirror attribute to create as many snapshot mirrors as you need for the snapshot volume. For a backup, you should usually only require the default of one.

It is also possible to make a snapshot plex from an existing plex in a volume. See "Converting a plex into a snapshot plex" on page 351 for details.

2 Choose a suitable time to create a snapshot. If possible, plan to take the snapshot at a time when users are accessing the volume as little as possible.

3 Create a snapshot volume using the following command:

```
# vxassist [-g diskgroup] snapshot [nmirror=N] volume snapshot
```

If required, use the `nmirror` attribute to specify the number of mirrors in the snapshot volume.

For example, to create a snapshot of `voldef`, use the following command:

```
# vxassist [-g diskgroup] snapshot voldef snapvol
```

The `vxassist snapshot` task detaches the finished snapshot mirror, creates a new volume, and attaches the snapshot mirror to it. This step should only take a few minutes. The snapshot volume, which reflects the original volume at the time of the snapshot, is now available for backing up, while the original volume continues to be available for applications and users.

If required, you can make snapshot volumes for several volumes in a disk group at the same time. See "Creating multiple snapshots" on page 352 for more information.

4 Use `fsck` (or some utility appropriate for the application running on the volume) to clean the temporary volume's contents. For example, you can use this command with a VxFS file system:

```
# fsck -F vxfs /dev/vx/rdsk/diskgroup/snapshot
```

5 If you require a backup of the data in the snapshot, use an appropriate utility or operating system command to copy the contents of the snapshot to tape, or to some other backup medium.

When the backup is complete, you have the following choices for what to do with the snapshot volume:

- Reattach some or all of the plexes of the snapshot volume with the original volume as described in "Reattaching a snapshot volume" on page 352. If FastResync was enabled on the volume before the snapshot was taken, this speeds resynchronization of the snapshot plexes before the backup cycle starts again at step 3.
- Dissociate the snapshot volume entirely from the original volume as described in "Dissociating a snapshot volume" on page 354. This may be useful if you want to use the copy for other purposes such as testing or report generation.
- Remove the snapshot volume to save space with this command:

```
# vxedit [-g diskgroup] -rf rm snapshot
```

Note: Dissociating or removing the snapshot volume loses the advantage of fast resynchronization if FastResync was enabled. If there are no further snapshot plexes available, any subsequent snapshots that you take require another complete copy of the original volume to be made.

Converting a plex into a snapshot plex

Note: The procedure described in this section cannot be used with layered volumes or any volume that has an associated version 20 DCO volume.

It is recommended that the instant snapshot feature is used in preference to the procedure described in this section.

In some circumstances, you may find it more convenient to convert an existing plex in a volume into a snapshot plex rather than running `vxassist snapstart`. For example, you may want to do this if you are short of disk space for creating the snapshot plex and the volume that you want to snapshot contains more than two plexes.

The procedure can also be used to speed up the creation of a snapshot volume when a mirrored volume is created with more than two plexes and `init=active` is specified.

Note: It is advisable to retain at least two plexes in a volume to maintain data redundancy.

To convert an existing plex into a snapshot plex for a volume on which Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] -o dcoplex=dcologplex convert \
    state=SNAPDONE plex
```

dcologplex is the name of an existing DCO plex that is to be associated with the new snapshot plex. You can use the `vxprint` command to find out the name of the DCO volume as described in "Adding a version 0 DCO and DCO volume" on page 356.

For example, to make a snapshot plex from the plex `trivol-03` in the 3-plex volume `trivol`, you would use the following command:

```
# vxplex -o dcoplex=trivol_dco-03 convert state=SNAPDONE \
  trivol-03
```

Here the DCO plex `trivol_dco_03` is specified as the DCO plex for the new snapshot plex.

To convert an existing plex into a snapshot plex in the SNAPDONE state for a volume on which Non-Persistent FastResync is enabled, use the following command:

```
# vxplex [-g diskgroup] convert state=SNAPDONE plex
```

A converted plex is in the SNAPDONE state, and can be used immediately to create a snapshot volume.

Note: The last complete regular plex in a volume, an incomplete regular plex, or a dirty region logging (DRL) log plex cannot be converted into a snapshot plex.

Creating multiple snapshots

To make it easier to create snapshots of several volumes at the same time, the snapshot option accepts more than one volume name as its argument, for example:

```
# vxassist [-g diskgroup] snapshot volume1 volume2 ...
```

By default, the first snapshot volume is named `SNAP-`*volume*, and each subsequent snapshot is named `SNAP`*number*`-`*volume*, where *number* is a unique serial number, and *volume* is the name of the volume for which the snapshot is being taken. This default pattern can be overridden by using the option `-o name=`*pattern*, as described on the `vxassist`(1M) manual page. For example, the pattern `SNAP%v-%d` reverses the order of the *number* and *volume* components in the name.

To snapshot all the volumes in a single disk group, specify the option `-o allvols` to `vxassist`:

```
# vxassist -g diskgroup -o allvols snapshot
```

This operation requires that all `snapstart` operations are complete on the volumes. It fails if any of the volumes in the disk group do not have a complete snapshot plex in the SNAPDONE state.

Reattaching a snapshot volume

Note: The information in this section does not apply to RAID-5 volumes unless they have been converted to a special layered volume layout by the addition of a DCO and DCO volume. See “Adding a version 0 DCO and DCO volume” on page 356 for details.

Snapback merges a snapshot copy of a volume with the original volume. One or more snapshot plexes are detached from the snapshot volume and re-attached to the original volume. The snapshot volume is removed if all its snapshot

plexes are snapped back. This task resynchronizes the data in the volume so that the plexes are consistent.

Note: To enhance the efficiency of the snapback operation, enable FastResync on the volume before taking the snapshot, as described in "Enabling FastResync on a volume" on page 292.

To merge one snapshot plex with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback snapshot
```

where *snapshot* is the snapshot copy of the volume.

To merge all snapshot plexes in the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] -o allplexes snapback snapshot
```

To merge a specified number of plexes from the snapshot volume with the original volume, use the following command:

```
# vxassist [-g diskgroup] snapback nmirror=number snapshot
```

Here the `nmirror` attribute specifies the number of mirrors in the snapshot volume that are to be re-attached.

Once the snapshot plexes have been reattached and their data resynchronized, they are ready to be used in another `snapshot` operation.

By default, the data in the original volume is used to update the snapshot plexes that have been re-attached. To copy the data from the replica volume instead, use the following command:

```
# vxassist [-g diskgroup] -o resyncfromreplica snapback snapshot
```

Caution: Always unmount the snapshot volume (if mounted) before performing a snapback. In addition, you must unmount the file system corresponding to the primary volume before using the `resyncfromreplica` option.

Adding plexes to a snapshot volume

If you want to retain the existing plexes in a snapshot volume after a snapback operation, you can create additional snapshot plexes that are to be used for the snapback.

To add plexes to a snapshot volume

1 Use the following `vxprint` commands to discover the names of the snapshot volume's data change object (DCO) and DCO volume:

```
# DCONAME=`vxprint [-g diskgroup] -F%dco_name snapshot`
# DCOVOL=`vxprint [-g diskgroup] -F%log_vol $DCONAME`
```

2 Use the `vxassist mirror` command to create mirrors of the existing snapshot volume and its DCO volume:

```
# vxassist -g diskgroup mirror snapshot
# vxassist -g diskgroup mirror $DCOVOL
```

Note: The new plex in the DCO volume is required for use with the new data plex in the snapshot.

3 Use the `vxprint` command to find out the name of the additional snapshot plex:

```
# vxprint -g diskgroup snapshot
```

4 Use the `vxprint` command to find out the record ID of the additional DCO plex:

```
# vxprint -g diskgroup -F%rid $DCOVOL
```

5 Use the `vxedit` command to set the `dco_plex_rid` field of the new data plex to the name of the new DCO plex:

```
# vxedit -g diskgroup set dco_plex_rid=dco_plex_rid new_plex
```

The new data plex is now ready to be used to perform a snapback operation.

Dissociating a snapshot volume

The link between a snapshot and its original volume can be permanently broken so that the snapshot volume becomes an independent volume. Use the following command to dissociate the snapshot volume, *snapshot*:

```
# vxassist snapclear snapshot
```

Displaying snapshot information

The `vxassist snapprint` command displays the associations between the original volumes and their respective replicas (snapshot copies):

```
# vxassist snapprint [volume]
```

Output from this command is shown in the following examples:

```
# vxassist -g mydg snapprint v1
V  NAME          USETYPE      LENGTH
SS SNAPOBJ       NAME         LENGTH     %DIRTY
DP NAME          VOLUME       LENGTH     %DIRTY

v  v1            fsgen        20480
ss SNAP-v1_snp   SNAP-v1      20480      4
dp v1-01         v1           20480      0
dp v1-02         v1           20480      0

v  SNAP-v1       fsgen        20480
ss v1_snp        v1           20480      0
# vxassist -g mydg snapprint v2
V  NAME          USETYPE      LENGTH
SS SNAPOBJ       NAME         LENGTH     %DIRTY
DP NAME          VOLUME       LENGTH     %DIRTY

v  v2            fsgen        20480
ss --            SNAP-v2      20480      0
dp v2-01         v2           20480      0

v  SNAP-v2       fsgen        20480
ss --            v2           20480      0
```

In this example, Persistent FastResync is enabled on volume `v1`, and Non-Persistent FastResync on volume `v2`. Lines beginning with `v`, `dp` and `ss` indicate a volume, detached plex and snapshot plex respectively. The `%DIRTY` field indicates the percentage of a snapshot plex or detached plex that is dirty with respect to the original volume. Notice that no snap objects are associated with volume `v2` or with its snapshot volume `SNAP-v2`. See "How persistent FastResync works with snapshots" on page 70 for more information about snap objects.

If a volume is specified, the `snapprint` command displays an error message if no FastResync maps are enabled for that volume.

Adding a version 0 DCO and DCO volume

Note: The procedure described in this section adds a DCO log volume that has a version 0 layout as introduced in VxVM 3.2. The version 0 layout supports traditional (third-mirror break-off) snapshots, but not full-sized or space-optimized instant snapshots. See "Version 0 DCO volume layout" on page 69 and "Version 20 DCO volume layout" on page 69 for a description of the differences between the old and new DCO volume layouts.

See "Determining the DCO version number" on page 277 for details of how to determine the version number of a volume's DCO.

To put Persistent FastResync into effect for a volume, a Data Change Object (DCO) and DCO volume must first be associated with that volume. When you have added a DCO object and DCO volume to a volume, you can then enable Persistent FastResync on the volume as described in "Enabling FastResync on a volume" on page 292.

Note: You need a Veritas FlashSnap™ or FastResync license to use the FastResync feature. Even if you do not have a license, you can configure a DCO object and DCO volume so that snap objects are associated with the original and snapshot volumes. For more information about snap objects, see "How persistent FastResync works with snapshots" on page 70.

To add a DCO object and DCO volume to an existing volume (which may already have dirty region logging (DRL) enabled), use the following procedure:

1. Ensure that the disk group containing the existing volume has been upgraded to at least version 90. Use the following command to check the version of a disk group:

   ```
   # vxdg list diskgroup
   ```

 To upgrade a disk group to the latest version, use the following command:

   ```
   # vxdg upgrade diskgroup
   ```

 For more information, see "Upgrading a disk group" on page 208.

2. Use the following command to turn off Non-Persistent FastResync on the original volume if it is currently enabled:

   ```
   # vxvol [-g diskgroup] set fastresync=off volume
   ```

 If you are uncertain about which volumes have Non-Persistent FastResync enabled, use the following command to obtain a listing of such volumes:

   ```
   # vxprint [-g diskgroup] -F "%name" \
     -e "v_fastresync=on && !v_hasdcolog"
   ```

3 Use the following command to add a DCO and DCO volume to the existing volume:

```
# vxassist [-g diskgroup] addlog volume logtype=dco \
  [ndcomirror=number] [dcolen=size] [storage_attributes]
```

For non-layered volumes, the default number of plexes in the mirrored DCO volume is equal to the lesser of the number of plexes in the data volume or 2. For layered volumes, the default number of DCO plexes is always 2. If required, use the `ndcomirror` attribute to specify a different *number*. It is recommended that you configure as many DCO plexes as there are existing data and snapshot plexes in the volume. For example, specify `ndcomirror=3` when adding a DCO to a 3-way mirrored volume.

The default size of each plex is 132 blocks. You can use the `dcolen` attribute to specify a different *size*. If specified, the size of the plex must be an integer multiple of 33 blocks from 33 up to a maximum of 2112 blocks.

You can specify `vxassist`-style storage attributes to define the disks that can and/or cannot be used for the plexes of the DCO volume. See "Specifying storage for version 0 DCO plexes" on page 357 for details.

Specifying storage for version 0 DCO plexes

Note: The operations in this section relate to version 0 DCO volumes. They are not supported for the version 20 DCO volume layout that was introduced in VxVM 4.0.

If the disks that contain volumes and their snapshots are to be moved or split into different disk groups, the disks that contain their respective DCO plexes must be able to accompany them. By default, VxVM attempts to place the DCO plexes on the same disks as the data plexes of the parent volume. However, this may be impossible if there is insufficient space available on those disks. In this case, VxVM uses any available space on other disks in the disk group. If the DCO plexes are placed on disks which are used to hold the plexes of other volumes, this may cause problems when you subsequently attempt to move volumes into other disk groups.

You can use storage attributes to specify explicitly which disks to use for the DCO plexes. If possible, specify the same disks as those on which the volume is configured. For example, to add a DCO object and DCO volume with plexes on `mydg05` and `mydg06`, and a plex size of 264 blocks to the volume, `myvol`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg addlog myvol logtype=dco dcolen=264 \
  mydg05 mydg06
```

To view the details of the DCO object and DCO volume that are associated with a volume, use the `vxprint` command. The following is partial `vxprint` output for

the volume named vol1 (the TUTIL0 and PUTIL0 columns are omitted for clarity):

```
TY  NAME         ASSOC        KSTATE   LENGTH  PLOFFS   STATE  ...
v   vol1         fsgen        ENABLED  1024    -        ACTIVE
pl  vol1-01      vol1         ENABLED  1024    -        ACTIVE
sd  disk01-01    vol1-01      ENABLED  1024    0        -
pl  vol1-02      vol1         ENABLED  1024    -        ACTIVE
sd  disk02-01    vol1-02      ENABLED  1024    0        -
dc  vol1_dco     vol1         -        -       -        -
v   vol1_dcl     gen          ENABLED  132     -        ACTIVE
pl  vol1_dcl-01  vol1_dcl     ENABLED  132     -        ACTIVE
sd  disk03-01    vol1_dcl-01  ENABLED  132     0        -
pl  vol1_dcl-02  vol1_dcl     ENABLED  132     -        ACTIVE
sd  disk04-01    vol1_dcl-02  ENABLED  132       0        -
```

In this output, the DCO object is shown as vol1_dco, and the DCO volume as vol1_dcl with 2 plexes, vol1_dcl-01 and vol1_dcl-02.

If required, you can use the vxassist move command to relocate DCO plexes to different disks. For example, the following command moves the plexes of the DCO volume, vol1_dcl, for volume vol1 from disk03 and disk04 to disk07 and disk08:

```
# vxassist -g mydg move vol1_dcl !disk03 !disk04 disk07 disk08
```

For more information, see "Moving DCO volumes between disk groups" on page 200, and the vxassist(1M) manual page.

Removing a version 0 DCO and DCO volume

Note: The operations in this section relate to version 0 DCO volumes. They are not supported for the version 20 DCO volume layout that was introduced in VxVM 4.0.

To dissociate a DCO object, DCO volume and any snap objects from a volume, use the following command:

```
# vxassist [-g diskgroup] remove log volume logtype=dco
```

This completely removes the DCO object, DCO volume and any snap objects. It also has the effect of disabling FastResync for the volume.

Alternatively, you can use the vxdco command to the same effect:

```
# vxdco [-g diskgroup] [-o rm] dis dco_obj
```

The default name of the DCO object, *dco_obj*, for a volume is usually formed by appending the string _dco to the name of the parent volume. To find out the name of the associated DCO object, use the vxprint command on the volume.

To dissociate, but not remove, the DCO object, DCO volume and any snap objects from the volume, myvol, in the disk group, mydg, use the following command:

```
# vxdco -g mydg dis myvol_dco
```

This form of the command dissociates the DCO object from the volume but does not destroy it or the DCO volume. If the `-o rm` option is specified, the DCO object, DCO volume and its plexes, and any snap objects are also removed.

Note: Dissociating a DCO and DCO volume disables Persistent FastResync on the volume. A full resynchronization of any remaining snapshots is required when they are snapped back.

For more information, see the `vxassist`(1M) and `vxdco`(1M) manual pages.

Reattaching a version 0 DCO and DCO volume

Note: The operations in this section relate to version 0 DCO volumes. They are not supported for version 20 DCO volume layout that was introduced in VxVM 4.0.

If the DCO object and DCO volume are not removed by specifying the `-o rm` option to `vxdco`, they can be reattached to the parent volume using the following command:

```
# vxdco [-g diskgroup] att volume dco_obj
```

For example, to reattach the DCO object, `myvol_dco`, to the volume, `myvol`, use the following command:

```
# vxdco -g mydg att myvol myvol_dco
```

For more information, see the `vxdco`(1M) manual page.

Chapter 10

Creating and administering volume sets

This chapter describes how to use the `vxvset` command to create and administer volume sets in Veritas Volume Manager (VxVM). Volume sets enable the use of the Multi-Volume Support feature with Veritas File System (VxFS). It is also possible to use the Veritas Enterprise Administrator (VEA) to create and administer volumes sets. For more information, see the VEA online help.

For full details of the usage of the `vxvset` command, see the vxvset(1M) manual page.

Note: Most VxVM commands require superuser or equivalent privileges.

Please note the following limitation of volume sets:

- A maximum of 2048 volumes may be configured in a volume set.
- Only Veritas File System is supported on a volume set.
- The first volume (index 0) in a volume set must be larger than the sum of the total volume size divided by 4000, the size of the VxFS intent log, and 1MB.
- Raw I/O from and to a volume set is not supported.
- Raw I/O from and to the component volumes of a volume set is supported under certain conditions. See “Raw device node access to component volumes” on page 364 for more information.
- Resizing a volume setwith an unmounted file system is not supported.

- Volume sets can be used in place of volumes with the following `vxsnap` operations on instant snapshots: `addmir`, `dis`, `make`, `prepare`, `reattach`, `refresh`, `restore`, `rmmir`, `split`, `syncpause`, `syncresume`, `syncstart`, `syncstop`, `syncwait`, and `unprepare`. The third-mirror break-off usage model for full-sized instant snapshots is supported for volume sets provided that sufficient plexes exist for each volume in the volume set. See "Administering volume snapshots" on page 303 and "Creating instant snapshots of volume sets" on page 334 for more information.
- A full-sized snapshot of a volume set must itself be a volume set with the same number of volumes and the same volume index numbers as the parent. The corresponding volumes in the parent and snapshot volume sets are also subject to the same restrictions as apply between standalone volumes and their snapshots.

Creating a volume set

To create a volume set for use by Veritas File System (VxFS), use the following command:

```
# vxvset [-g diskgroup] -t vxfs make volset volume
```

Here *volset* is the name of the volume set, and *volume* is the name of the first volume in the volume set. The `-t` option defines the content handler subdirectory for the application that is to be used with the volume. This subdirectory contains utilities that an application uses to operate on the volume set. As the operation of these utilities is determined by the requirements of the application and not by VxVM, it is not discussed further here.

For example, to create a volume set named `myvset` that contains the volume `vol1`, in the disk group `mydg`, you would use the following command:

```
# vxvset -g mydg -t vxfs make myvset vol1
```

Adding a volume to a volume set

Having created a volume set containing a single volume, you can use the following command to add further volumes to the volume set:

```
# vxvset [-g diskgroup] [-f] addvol volset volume
```

For example, to add the volume `vol2`, to the volume set `myvset`, use the following command:

```
# vxvset -g mydg addvol myvset vol2
```

Caution: The `-f` (force) option must be specified if the volume being added, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

Listing details of volume sets

To list the details of the component volumes of a volume set, use the following command:

```
# vxvset [-g diskgroup] list [volset]
```

If the name of a volume set is not specified, the command lists the details of all volume sets in a disk group, as shown in the following example:

```
# vxvset -g mydg list
NAME        GROUP       NVOLS       CONTEXT
set1        mydg            3       -
set2        mydg            2       -
```

To list the details of each volume in a volume set, specify the name of the volume set as an argument to the command:

```
# vxvset -g mydg list set1
VOLUME            INDEX         LENGTH      KSTATE   CONTEXT
vol1                  0       12582912     ENABLED   -
vol2                  1       12582912     ENABLED   -
vol3                  2       12582912     ENABLED   -
```

The context field contains details of any string that the application has set up for the volume or volume set to tag its purpose.

Stopping and starting volume sets

Under some circumstances, you may need to stop and restart a volume set. For example, a volume within the set may have become detached, as shown here:

```
# vxvset -g mydg list set1
VOLUME            INDEX         LENGTH      KSTATE   CONTEXT
vol1                  0       12582912   DETACHED   -
vol2                  1       12582912    ENABLED   -
vol3                  2       12582912    ENABLED   -
```

To stop and restart one or more volume sets, use the following commands:

```
# vxvset [-g diskgroup] stop volset ...
# vxvset [-g diskgroup] start volset ...
```

For the example given previously, the effect of running these commands on the component volumes is shown below:

```
# vxvset -g mydg stop set1
```

```
# vxvset -g mydg list set1
VOLUME            INDEX        LENGTH     KSTATE  CONTEXT
vol1                  0      12582912   DISABLED  -
vol2                  1      12582912   DISABLED  -
vol3                  2      12582912   DISABLED  -

# vxvset -g mydg start set1

# vxvset -g mydg list set1
VOLUME            INDEX        LENGTH     KSTATE  CONTEXT
vol1                  0      12582912    ENABLED  -
vol2                  1      12582912    ENABLED  -
vol3                  2      12582912    ENABLED  -
```

Removing a volume from a volume set

To remove a component volume from a volume set, use the following command:

```
# vxvset [-g diskgroup] [-f] rmvol volset volume
```

For example, the following commands remove the volumes, `vol1` and `vol2`, from the volume set `myvset`:

```
# vxvset -g mydg rmvol myvset vol1
# vxvset -g mydg rmvol myvset vol2
```

Note: When the final volume is removed, this deletes the volume set.

Caution: The `-f` (force) option must be specified if the volume being removed, or any volume in the volume set, is either a snapshot or the parent of a snapshot. Using this option can potentially cause inconsistencies in a snapshot hierarchy if any of the volumes involved in the operation is already in a snapshot chain.

Raw device node access to component volumes

To guard against accidental file system and data corruption, the device nodes of the component volumes are configured by default not to have raw and block entries in the `/dev/vx/rdsk/diskgroup` and `/dev/vx/dsk/diskgroup` directories. As a result, applications are prevented from directly reading from or writing to the component volumes of a volume set.

If some applications, such as the raw volume backup and restore feature of the Veritas NetBackup™ software, need to read from or write to the component volumes by accessing raw device nodes in the `/dev/vx/rdsk/diskgroup` directory, this is supported by specifying additional command-line options to the `vxvset` command. Access to the block device nodes of the component volumes of a volume set is unsupported.

Caution: Writing directly to or reading from the raw device node of a component volume of a volume set should only be performed if it is known that the volume's data will not otherwise change during the period of access.

All of the raw device nodes for the component volumes of a volume set can be created or removed in a single operation. Raw device nodes for any volumes added to a volume set are created automatically as required, and inherit the access mode of the existing device nodes.

Access to the raw device nodes for the component volumes can be configured to be read-only or read-write. This mode is shared by all the raw device nodes for the component volumes of a volume set. The read-only access mode implies that any writes to the raw device will fail, however writes using the `ioctl` interface or by VxFS to update metadata are not prevented. The read-write access mode allows direct writes via the raw device. The access mode to the raw device nodes of a volume set can be changed as required.

The presence of raw device nodes and their access mode is persistent across system reboots.

Note the following limitations of this feature:

- The disk group version must be 120 or greater.
- Access to the raw device nodes of the component volumes of a volume set is only supported for private disk groups; it is not supported for shared disk groups in a cluster.

Enabling raw device access when creating a volume set

To enable raw device access when creating a volume set, use the following form of the `vxvset make` command:

```
# vxvset [-g diskgroup] -o makedev=on \
  [-o compvol_access={read-only|read-write}] \
  [-o index] [-c "ch_addopt"] make vset vol [index]
```

The `-o makedev=on` option enables the creation of raw device nodes for the component volumes at the same time that the volume set is created. The default is setting is `off`.

If the `-o compvol_access=read-write` option is specified, direct writes are allowed to the raw device of each component volume. If the value is set to `read-only`, only reads are allowed from the raw device of each component volume.

If the `-o makedev=on` option is specified, but `-o compvol_access` is not specified, the default access mode is `read-only`.

If the `vxvset addvol` command is subsequently used to add a volume to a volume set, a new raw device node is created in `/dev/vx/rdsk/diskgroup` if the

value of the makedev attribute is currently set to on. The access mode is determined by the current setting of the compvol_access attribute.

The following example creates a volume set, myvset1, containing the volume, myvol1, in the disk group, mydg, with raw device access enabled in read-write mode:

```
# vxvset -g mydg -o makedev=on -o compvol_access=read-write \
  make myvset1 myvol1
```

Displaying the raw device access settings for a volume set

You can use the vxprint -m command to display the current settings for a volume set. If the makedev attribute is set to on, one of the following is displayed in the output:

vset_devinfo=on:read-only Raw device nodes in read-only mode.

vset_devinfo=on:read-write Raw device nodes in read-write mode.

This field is not displayed if makedev is set to off.

Note: If the output from the vxprint -m command is fed to the vxmake command to recreate a volume set, the vset_devinfo attribute must set to off. Use the vxvset set command to re-enable raw device access with the desired access mode as described in "Controlling raw device access for an existing volume set" on page 366.

Controlling raw device access for an existing volume set

To enable or disable raw device node access for an existing volume set, use the following command:

```
# vxvset [-g diskgroup] [-f] set makedev={on|off} vset
```

The makedev attribute can be specified to the vxvset set command to create (makedev=on) or remove (makedev=off) the raw device nodes for the component volumes of a volume set. If any of the component volumes are open, the -f (force) option must be specified to set the attribute to off.

Note: Specifying makedev=off removes the existing raw device nodes from the /dev/vx/rdsk/*diskgroup* directory.

If the makedev attribute is set to off, and you use the mknod command to create the raw device nodes, you cannot read from or write to those nodes unless you set the value of makedev to on.

The syntax for setting the compvol_access attribute on a volume set is:

```
# vxvset [-g diskgroup] [-f] set \
  compvol_access={read-only|read-write} vset
```

The compvol_access attribute can be specified to the vxvset set command to change the access mode to the component volumes of a volume set. If any of the component volumes are open, the -f (force) option must be specified to set the attribute to read-only.

The following example sets the makedev=on and compvol_access=read-only attributes on a volume set, myvset2, in the disk group, mydg:

```
# vxvset -g mydg set makedev=on myvset2
```

The next example sets the compvol_access=read-write attribute on the volume set, myvset2:

```
# vxvset -g mydg set compvol_access=read-write myvset2
```

The final example removes raw device node access for the volume set, myvset2:

```
# vxvset -g mydg set makedev=off myvset2
```

Chapter 11

Configuring off-host processing

Off-host processing allows you to implement the following activities:

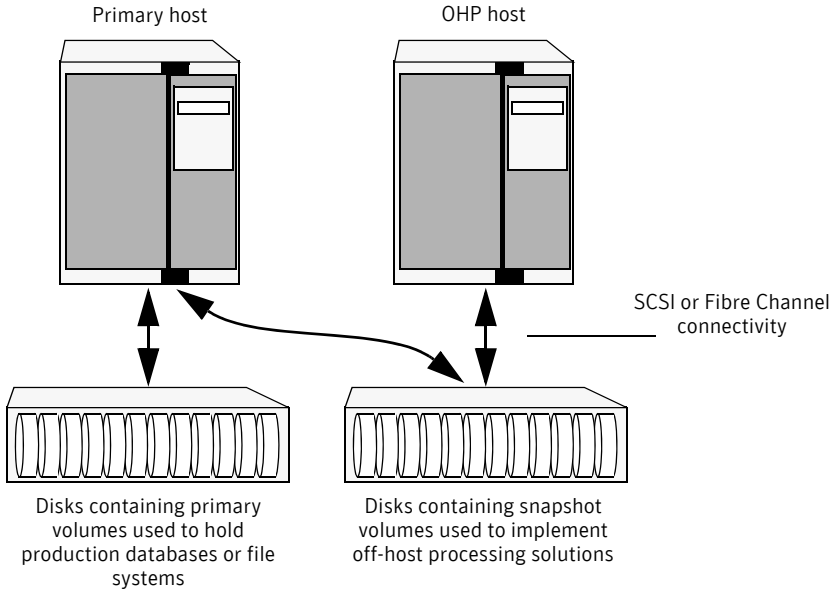
Data backup	As the requirement for 24 x 7 availability becomes essential for many businesses, organizations cannot afford the downtime involved in backing up critical data offline. By taking a snapshot of the data, and backing up from this snapshot, business-critical applications can continue to run without extended down time or impacted performance.
Decision support analysis and reporting	Because snapshots hold a point-in-time copy of a production database, a replica of the database can be set up using the snapshots. Operations such as decision support analysis and business reporting do not require access to up-to-the-minute information. This means that they can use a database copy that is running on a host other than the primary. When required, the database copy can quickly be synchronized with the data in the primary database.
Testing and training	Development or service groups can use snapshots as test data for new applications. Snapshot data provides developers, system testers and QA groups with a realistic basis for testing the robustness, integrity and performance of new applications.
Database error recovery	Logic errors caused by an administrator or an application program can compromise the integrity of a database. By restoring the database table files from a snapshot copy, the database can be recovered more quickly than by full restoration from tape or other backup media.

Off-host processing is made simpler by using linked break-off snapshots, which are described in “Linked break-off snapshot volumes” on page 311.

Implementing off-host processing solutions

As shown in Figure 11-1, by accessing snapshot volumes from a lightly loaded host (shown here as the off-host processing (OHP) host), CPU- and I/O-intensive operations for online backup and decision support do not degrade the performance of the primary host that is performing the main production activity (such as running a database). Also, if you place the snapshot volumes on disks that are attached to different host controllers than the disks in the primary volumes, it is possible to avoid contending with the primary host for I/O resources.

Figure 11-1 Example implementation of off-host processing



The following sections describe how you can apply off-host processing to implement regular online backup of a volume in a private disk group, and to set up a replica of a production database for decision support. Two applications are outlined in the following sections:

- Implementing off-host online backup

- Implementing decision support

These applications use the Persistent FastResync feature of VxVM in conjunction with linked break-off snapshots.

Note: A volume snapshot represents the data that exists in a volume at a given point in time. As such, VxVM does not have any knowledge of data that is cached by the overlying file system, or by applications such as databases that have files open in the file system. If the `fsgen` volume usage type is set on a volume that contains a Veritas File System (VxFS), intent logging of the file system metadata ensures the internal consistency of the file system that is backed up. For other file system types, depending on the intent logging capabilities of the file system, there may potentially be inconsistencies between in-memory data and the data in the snapshot image.

For databases, a suitable mechanism must additionally be used to ensure the integrity of tablespace data when the volume snapshot is taken. The facility to temporarily suspend file system I/O is provided by most modern database software. For ordinary files in a file system, which may be open to a wide variety of different applications, there may be no way to ensure the complete integrity of the file data other than by shutting down the applications and temporarily unmounting the file system. In many cases, it may only be important to ensure the integrity of file data that is not in active use at the time that you take the snapshot.

Implementing off-host online backup

This section describes a procedure for implementing off-host online backup for a volume in a private disk group. The intention is to present an outline of how to set up a regular backup cycle. It is beyond the scope of this guide to describe how to configure a database to use this procedure, or how to perform the backup itself.

To back up a volume in a private disk group

1 Use the following command on the primary host to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume:

```
# vxprint -g volumedg -F%instant volume
```

This command returns `on` if the volume can be used for instant snapshot operations; otherwise, it returns `off`.

Note: If the volume was created under VxVM 4.0 or a later release, and it is not associated with a new-style DCO object and DCO volume, follow the procedure described in "Preparing a volume for DRL and instant snapshots" on page 275.

If the volume was created before release 4.0 of VxVM, and has any attached snapshot plexes, or is associated with any snapshot volumes, follow the procedure given in "Upgrading existing volumes to use version 20 DCOs" on page 279.

2 Use the following command on the primary host to check whether FastResync is enabled on the volume:

```
# vxprint -g volumedg -F%fastresync volume
```

This command returns `on` if FastResync is enabled; otherwise, it returns `off`.

If FastResync is disabled, enable it using the following command on the primary host:

```
# vxvol -g volumedg set fastresync=on volume
```

3 On the primary host, create a new volume in a separate disk group for use as the snapshot volume as described in "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 323. It is recommended that a snapshot disk group is dedicated to maintaining only those disks that are used for off-host processing.

4 On the primary host, link the snapshot volume in the snapshot disk group to the data volume:

```
# vxsnap -g volumedg -b addmir volume mirvol=snapvol \
  mirdg=snapvoldg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol \
  mirdg=snapvoldg
```

Note: This step sets up the snapshot volumes. When you are ready to create a backup, proceed to step 5.

5 On the primary host, suspend updates to the volume that contains the database tables. A database may have a hot backup mode that allows you to do this by temporarily suspending writes to its tables.

6 Create the snapshot volume, *snapvol*, by running the following command on the primary host:

```
# vxsnap -g volumedg make \
  source=volume/snapvol=snapvol/snapdg=snapvoldg
```

If a database spans more than one volume, you can specify all the volumes and their snapshot volumes using one command, as shown in this example:

```
# vxsnap -g dbasedg make \
  source=vol1/snapvol=snapvol1/snapdg=sdg \
  source=vol2/snapvol=snapvol2/snapdg=sdg \
  source=vol3/snapvol=snapvol3/snapdg=sdg
```

This step sets up the snapshot volumes ready for the backup cycle, and starts tracking changes to the original volumes.

7 On the primary host, if you temporarily suspended updates to a volume in step 5, release all the database tables from hot backup mode.

8 On the primary host, deport the snapshot volume's disk group using the following command:

```
# vxdg deport snapvoldg
```

9 On the OHP host where the backup is to be performed, use the following command to import the snapshot volume's disk group:

```
# vxdg import snapvoldg
```

10 The snapshot volume is initially disabled following the join. Use the following commands on the OHP host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
# vxvol -g snapvoldg start snapvol
```

11 On the OHP host, back up the snapshot volume. If you need to remount the file system in the volume to back it up, first run `fsck` on the volume. The following are sample commands for checking and mounting a file system:

```
# fsck -F vxfs /dev/vx/rdsk/snapvoldg/snapvol
# mount -F vxfs /dev/vx/dsk/snapvoldg/snapvol mount_point
```

Back up the file system at this point, and then use the following command to unmount it.

```
# umount mount_point
```

12 On the OHP host, use the following command to deport the snapshot volume's disk group:

```
# vxdg deport snapvoldg
```

13 On the primary host, re-import the snapshot volume's disk group using the following command:

```
# vxdg import snapvoldg
```

14 The snapshot volume is initially disabled following the join. Use the following command on the primary host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
```

15 On the primary host, reattach the snapshot volume to its original volume using the following command:

```
# vxsnap -g snapvoldg reattach snapvol source=vol \
  sourcedg=volumedg
```

For example, to reattach the snapshot volumes `svol1`, `svol2` and `svol3`:

```
# vxsnap -g sdg reattach svol1 \
  source=vol1 sourcedg=dbasedg \
  svol2 source=vol2 sourcedg=dbasedg \
  svol3 source=vol3 sourcedg=dbasedg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol
```

Repeat step 5 through step 15 each time that you need to back up the volume.

Implementing decision support

This section describes a procedure for implementing off-host decision support for a volume in a private disk group. The intention is to present an outline of how to set up a replica database. It is beyond the scope of this guide to describe how to configure a database to use this procedure, or how to perform the decision support processing itself.

To set up a replica database using the table files that are configured within a volume in a private disk group

1. Use the following command on the primary host to see if the volume is associated with a version 20 data change object (DCO) and DCO volume that allow instant snapshots and Persistent FastResync to be used with the volume:

   ```
   # vxprint -g volumedg -F%instant volume
   ```

 This command returns `on` if the volume can be used for instant snapshot operations; otherwise, it returns `off`.

 Note: If the volume was created under VxVM 4.0 or a later release, and it is not associated with a new-style DCO object and DCO volume, follow the procedure described in "Preparing a volume for DRL and instant snapshots" on page 275.

 If the volume was created before release 4.0 of VxVM, and has any attached snapshot plexes, or is associated with any snapshot volumes, follow the procedure given in "Upgrading existing volumes to use version 20 DCOs" on page 279.

2. Use the following command on the primary host to check whether FastResync is enabled on a volume:

   ```
   # vxprint -g volumedg -F%fastresync volume
   ```

This command returns `on` if FastResync is enabled; otherwise, it returns `off`.

If FastResync is disabled, enable it using the following command on the primary host:

```
# vxvol -g volumedg set fastresync=on volume
```

3 Prepare the OHP host to receive the snapshot volume that contains the copy of the database tables. This may involve setting up private volumes to contain any redo logs, and configuring any files that are used to initialize the database.

4 On the primary host, create a new volume in a separate disk group for use as the snapshot volume as described in "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 323. It is recommended that a snapshot disk group is dedicated to maintaining only those disks that are used for off-host processing.

5 On the primary host, link the snapshot volume in the snapshot disk group to the data volume:

```
# vxsnap -g volumedg -b addmir volume mirvol=snapvol \
  mirdg=snapvoldg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol \
  mirdg=snapvoldg
```

Note: This step sets up the snapshot volumes. When you are ready to create a replica database, proceed to step 6.

6 On the primary host, suspend updates to the volume that contains the database tables. A database may have a hot backup mode that allows you to do this by temporarily suspending writes to its tables.

7 Create the snapshot volume, *snapvol*, by running the following command on the primary host:

```
# vxsnap -g volumedg make \
  source=volume/snapvol=snapvol/snapdg=snapvoldg
```

If a database spans more than one volume, you can specify all the volumes and their snapshot volumes using one command, as shown in this example:

```
# vxsnap -g dbasedg make \
  source=vol1/snapvol=snapvol1/snapdg=sdg \
  source=vol2/snapvol=snapvol2/snapdg=sdg \
  source=vol3/snapvol=snapvol3/snapdg=sdg
```

This step makes the snapshot volumes ready for the decision-support processing cycle, and starts tracking changes to the original volumes.

8 On the primary host, if you temporarily suspended updates to a volume in step 6, release all the database tables from hot backup mode.

9 On the primary host, deport the snapshot volume's disk group using the following command:

```
# vxdg deport snapvoldg
```

10 On the OHP host where the replica database is to be set up, use the following command to import the snapshot volume's disk group:

```
# vxdg import snapvoldg
```

11 The snapshot volume is initially disabled following the join. Use the following command on the OHP host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
```

12 On the OHP host, check and mount the snapshot volume. The following are sample commands for checking and mounting a file system:

```
# fsck -F vxfs /dev/vx/rdsk/snapvoldg/snapvol
# mount -F vxfs /dev/vx/dsk/snapvoldg/snapvol \
  mount_point
```

13 On the OHP host, use the appropriate database commands to recover and start the replica database for its decision support role.

When you want to resynchronize the snapshot volume' s data with the primary database, you can reattach the snapshot volume to the original volume as follows:

1 On the OHP host, shut down the replica database, and use the following command to unmount the snapshot volume:

```
# umount mount_point
```

2 On the OHP host, use the following command to deport the snapshot volume's disk group:

```
# vxdg deport snapvoldg
```

3 On the primary host, re-import the snapshot volume's disk group using the following command:

```
# vxdg import snapvoldg
```

4 The snapshot volume is initially disabled following the join. Use the following command on the primary host to recover and restart the snapshot volume:

```
# vxrecover -g snapvoldg -m snapvol
```

5 On the primary host, reattach the snapshot volume to its original volume using the following command:

```
# vxsnap -g snapvoldg reattach snapvol source=vol \
  sourcedg=volumedg
```

For example, to reattach the snapshot volumes `svol1`, `svol2` and `svol3`:

```
# vxsnap -g sdg reattach svol1 \
  source=vol1 sourcedg=dbasedg \
  svol2 source=vol2 sourcedg=dbasedg \
  svol3 source=vol3 sourcedg=dbasedg
```

You can use the `vxsnap snapwait` command to wait for synchronization of the linked snapshot volume to complete:

```
# vxsnap -g volumedg snapwait volume mirvol=snapvol
```

You can then resume the procedure from step 6 on page 375.

Chapter 12

Administering hot-relocation

If a volume has a disk I/O failure (for example, the disk has an uncorrectable error), Veritas Volume Manager (VxVM) can detach the plex involved in the failure. I/O stops on that plex but continues on the remaining plexes of the volume.

If a disk fails completely, VxVM can detach the disk from its disk group. All plexes on the disk are disabled. If there are any unmirrored volumes on a disk when it is detached, those volumes are also disabled.

Note: Apparent disk failure may not be due to a fault in the physical disk media or the disk controller, but may instead be caused by a fault in an intermediate or ancillary component such as a cable, host bus adapter, or power supply.

The hot-relocation feature in VxVM automatically detects disk failures, and notifies the system administrator and other nominated users of the failures by electronic mail. Hot-relocation also attempts to use spare disks and free disk space to restore redundancy and to preserve access to mirrored and RAID-5 volumes. For more information, see the section, "How hot-relocation works" on page 380.

If hot-relocation is disabled or you miss the electronic mail, you can use the `vxprint` command or the graphical user interface to examine the status of the disks. You may also see driver error messages on the console or in the system messages file.

Failed disks must be removed and replaced manually as described in "Removing and replacing disks" on page 112.

For more information about recovering volumes and their data after hardware failure, see the *Veritas Volume Manager Troubleshooting Guide.*

How hot-relocation works

Hot-relocation allows a system to react automatically to I/O failures on redundant (mirrored or RAID-5) VxVM objects, and to restore redundancy and access to those objects. VxVM detects I/O failures on objects and relocates the affected subdisks to disks designated as spare disks or to free space within the disk group. VxVM then reconstructs the objects that existed before the failure and makes them redundant and accessible again.

When a partial disk failure occurs (that is, a failure affecting only some subdisks on a disk), redundant data on the failed portion of the disk is relocated. Existing volumes on the unaffected portions of the disk remain accessible.

Note: Hot-relocation is only performed for redundant (mirrored or RAID-5) subdisks on a failed disk. Non-redundant subdisks on a failed disk are not relocated, but the system administrator is notified of their failure.

Hot-relocation is enabled by default and takes effect without the intervention of the system administrator when a failure occurs.

The hot-relocation daemon, `vxrelocd`, detects and reacts to VxVM events that signify the following types of failures:

Disk failure	This is normally detected as a result of an I/O failure from a VxVM object. VxVM attempts to correct the error. If the error cannot be corrected, VxVM tries to access configuration information in the private region of the disk. If it cannot access the private region, it considers the disk failed.
Plex failure	This is normally detected as a result of an uncorrectable I/O error in the plex (which affects subdisks within the plex). For mirrored volumes, the plex is detached.
RAID-5 subdisk failure	This is normally detected as a result of an uncorrectable I/O error. The subdisk is detached.

When `vxrelocd` detects such a failure, it performs the following steps:

- `vxrelocd` informs the system administrator (and other nominated users) by electronic mail of the failure and which VxVM objects are affected.
 See “Partial disk failure mail messages” on page 383.
 See “Complete disk failure mail messages” on page 384.
 See “Modifying the behavior of hot-relocation” on page 395.
- `vxrelocd` next determines if any subdisks can be relocated. `vxrelocd` looks for suitable space on disks that have been reserved as hot-relocation

spares (marked `spare`) in the disk group where the failure occurred. It then relocates the subdisks to use this space.

- If no spare disks are available or additional space is needed, `vxrelocd` uses free space on disks in the same disk group, except those disks that have been excluded for hot-relocation use (marked `nohotuse`). When `vxrelocd` has relocated the subdisks, it reattaches each relocated subdisk to its plex.
- Finally, `vxrelocd` initiates appropriate recovery procedures. For example, recovery includes mirror resynchronization for mirrored volumes or data recovery for RAID-5 volumes. It also notifies the system administrator of the hot-relocation and recovery actions that have been taken.

If relocation is not possible, `vxrelocd` notifies the system administrator and takes no further action.

Note: Hot-relocation does not guarantee the same layout of data or the same performance after relocation. The system administrator can make configuration changes after hot-relocation occurs.

Relocation of failing subdisks is not possible in the following cases:

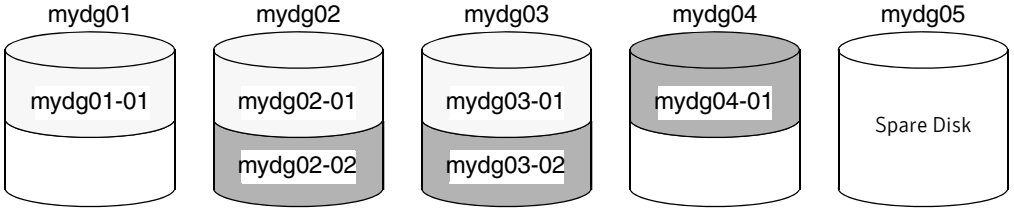
- The failing subdisks are on non-redundant volumes (that is, volumes of types other than mirrored or RAID-5).
- There are insufficient spare disks or free disk space in the disk group.
- The only available space is on a disk that already contains a mirror of the failing plex.
- The only available space is on a disk that already contains the RAID-5 log plex or one of its healthy subdisks. Failing subdisks in the RAID-5 plex cannot be relocated.
- If a mirrored volume has a dirty region logging (DRL) log subdisk as part of its data plex, failing subdisks belonging to that plex cannot be relocated.
- If a RAID-5 volume log plex or a mirrored volume DRL log plex fails, a new log plex is created elsewhere. There is no need to relocate the failed subdisks of the log plex.

See the `vxrelocd`(1M) manual page for more information about the hot-relocation daemon.

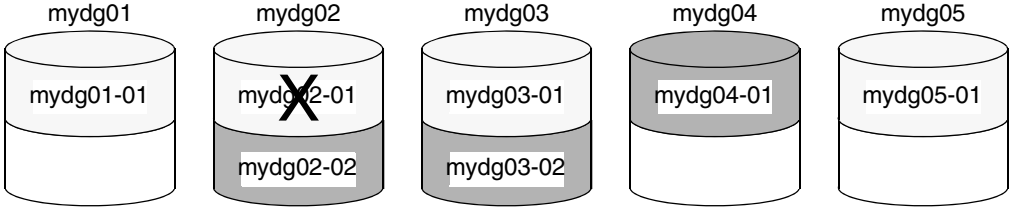
Figure 12-1 illustrates the hot-relocation process in the case of the failure of a single subdisk of a RAID-5 volume.

Figure 12-1 Example of hot-relocation for a subdisk in a RAID-5 volume

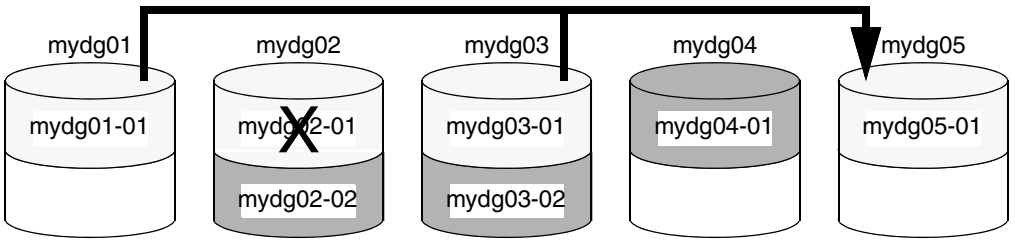
a) Disk group contains five disks. Two RAID-5 volumes are configured across four of the disks. One spare disk is available for hot-relocation.



b) Subdisk mydg02-01 in one RAID-5 volume fails. Hot-relocation replaces it with subdisk mydg05-01 that it has created on the spare disk, and then initiates recovery of the RAID-5 volume.



c) RAID-5 recovery recreates subdisk mydg02-01's data and parity on subdisk mydg05-01 from the data and parity information remaining on subdisks mydg01-01 and mydg03-01.



Partial disk failure mail messages

If hot-relocation is enabled when a plex or disk is detached by a failure, mail indicating the failed objects is sent to `root`. If a partial disk failure occurs, the mail identifies the failed plexes. For example, if a disk containing mirrored volumes fails, you can receive mail information as shown in the following example:

```
To: root
Subject: Volume Manager failures on host teal
Failures have been detected by the Veritas Volume Manager:

failed plexes:
home-02
src-02
```

See "Modifying the behavior of hot-relocation" on page 395 for information on how to send the mail to users other than `root`.

You can determine which disk is causing the failures in the above example message by using the following command:

```
# vxstat -g mydg -s -ff home-02 src-02
```

The `-s` option asks for information about individual subdisks, and the `-ff` option displays the number of failed read and write operations. The following output display is typical:

```
                          FAILED
TYP NAME                READS    WRITES
sd mydg01-04                0         0
sd mydg01-06                0         0
sd mydg02-03                1         0
sd mydg02-04                1         0
```

This example shows failures on reading from subdisks `mydg02-03` and `mydg02-04` of disk `mydg02`.

Hot-relocation automatically relocates the affected subdisks and initiates any necessary recovery procedures. However, if relocation is not possible or the hot-relocation feature is disabled, you must investigate the problem and attempt to recover the plexes. Errors can be caused by cabling failures, so check the cables connecting your disks to your system. If there are obvious problems, correct them and recover the plexes using the following command:

```
# vxrecover -b -g mydg home src
```

This starts recovery of the failed plexes in the background (the command prompt reappears before the operation completes). If an error message appears later, or if the plexes become detached again and there are no obvious cabling failures, replace the disk (see "Removing and replacing disks" on page 112).

Complete disk failure mail messages

If a disk fails completely and hot-relocation is enabled, the mail message lists the disk that failed and all plexes that use the disk. For example, you can receive mail as shown in this example display:

```
To: root
Subject: Volume Manager failures on host teal

Failures have been detected by the Veritas Volume Manager:

failed disks:
mydg02

failed plexes:
home-02
src-02
mkting-01

failing disks:
mydg02
```

This message shows that `mydg02` was detached by a failure. When a disk is detached, I/O cannot get to that disk. The plexes `home-02`, `src-02`, and `mkting-01` were also detached (probably because of the failure of the disk).

As described in "Partial disk failure mail messages" on page 383, the problem can be a cabling error. If the problem is not a cabling error, replace the disk (see "Removing and replacing disks" on page 112).

How space is chosen for relocation

A spare disk must be initialized and placed in a disk group as a spare before it can be used for replacement purposes. If no disks have been designated as spares when a failure occurs, VxVM automatically uses any available free space in the disk group in which the failure occurs. If there is not enough spare disk space, a combination of spare space and free space is used.

The free space used in hot-relocation must not have been excluded from hot-relocation use. Disks can be excluded from hot-relocation use by using `vxdiskadm`, `vxedit` or the Veritas Enterprise Administrator (VEA).

You can designate one or more disks as hot-relocation spares within each disk group. Disks can be designated as spares by using `vxdiskadm`, `vxedit`, or the VEA. Disks designated as spares do not participate in the free space model and should not have storage space allocated on them.

When selecting space for relocation, hot-relocation preserves the redundancy characteristics of the VxVM object to which the relocated subdisk belongs. For example, hot-relocation ensures that subdisks from a failed plex are not relocated to a disk containing a mirror of the failed plex. If redundancy cannot be preserved using any available spare disks and/or free space, hot-relocation

does not take place. If relocation is not possible, the system administrator is notified and no further action is taken.

From the eligible disks, hot-relocation attempts to use the disk that is “closest” to the failed disk. The value of “closeness” depends on the controller, target, and disk number of the failed disk. A disk on the same controller as the failed disk is closer than a disk on a different controller. A disk under the same target as the failed disk is closer than one on a different target.

Hot-relocation tries to move all subdisks from a failing drive to the same destination disk, if possible.

When hot-relocation takes place, the failed subdisk is removed from the configuration database, and VxVM ensures that the disk space used by the failed subdisk is not recycled as free space.

Configuring a system for hot-relocation

By designating spare disks and making free space on disks available for use by hot relocation, you can control how disk space is used for relocating subdisks in the event of a disk failure. If the combined free space and space on spare disks is not sufficient or does not meet the redundancy constraints, the subdisks are not relocated.

- To find out which disks are spares or are excluded from hot-relocation, see “Displaying spare disk information” on page 386.

You can prepare for hot-relocation by designating one or more disks per disk group as hot-relocation spares.

- To designate a disk as being a hot-relocation spare for a disk group, see “Marking a disk as a hot-relocation spare” on page 387.
- To remove a disk from use as a hot-relocation spare, see “Removing a disk from use as a hot-relocation spare” on page 388.

If no spares are available at the time of a failure or if there is not enough space on the spares, free space on disks in the same disk group as where the failure occurred is automatically used, unless it has been excluded from hot-relocation use.

- To exclude a disk from hot-relocation use, see “Excluding a disk from hot-relocation use” on page 388.
- To make a disk available for hot-relocation use, see “Making a disk available for hot-relocation use” on page 389.

Depending on the locations of the relocated subdisks, you can choose to move them elsewhere after hot-relocation occurs (see “Configuring hot-relocation to use only spare disks” on page 390).

After a successful relocation, remove and replace the failed disk as described in “Removing and replacing disks” on page 112).

Displaying spare disk information

Use the following command to display information about spare disks that are available for relocation:

```
# vxdg [-g diskgroup] spare
```

The following is example output:

```
GROUP DISK      DEVICE       TAG       OFFSET    LENGTH      FLAGS
mydg  mydg02    c0t2d0       c0t2d0    0         658007      s
```

Here `mydg02` is the only disk designated as a spare in the `mydg` disk group. The `LENGTH` field indicates how much spare space is currently available on `mydg02` for relocation.

The following commands can also be used to display information about disks that are currently designated as spares:

- `vxdisk list` lists disk information and displays spare disks with a `spare` flag.
- `vxprint` lists disk and other information and displays spare disks with a `SPARE` flag.
- The `list` menu item on the `vxdiskadm` main menu lists all disks including spare disks.

Marking a disk as a hot-relocation spare

Hot-relocation allows the system to react automatically to I/O failure by relocating redundant subdisks to other disks. Hot-relocation then restores the affected VxVM objects and data. If a disk has already been designated as a spare in the disk group, the subdisks from the failed disk are relocated to the spare disk. Otherwise, any suitable free space in the disk group is used.

To designate a disk as a hot-relocation spare, enter the following command:

```
# vxedit [-g diskgroup] set spare=on diskname
```

where *diskname* is the disk media name.

For example, to designate `mydg01` as a spare in the disk group, `mydg`, enter the following command:

```
# vxedit -g mydg set spare=on mydg01
```

You can use the `vxdisk list` command to confirm that this disk is now a spare; `mydg01` should be listed with a `spare` flag.

Any VM disk in this disk group can now use this disk as a spare in the event of a failure. If a disk fails, hot-relocation automatically occurs (if possible). You are notified of the failure and relocation through electronic mail. After successful relocation, you may want to replace the failed disk.

To use vxdiskadm to designate a disk as a hot-relocation spare

1. Select menu item `11 (Mark a disk as a spare for a disk group)` from the `vxdiskadm` main menu.
2. At the following prompt, enter a disk media name (such as `mydg01`):

   ```
   Menu: VolumeManager/Disk/MarkSpareDisk

   Use this operation to mark a disk as a spare for a disk group.
   This operation takes, as input, a disk name. This is the same
   name that you gave to the disk when you added the disk to the
   disk group.

   Enter disk name [<disk>,list,q,?] mydg01
   ```

 The following notice is displayed when the disk has been marked as spare:

   ```
   VxVM NOTICE V-5-2-219 Marking of mydg01 in mydg as a spare disk
   is complete.
   ```

3. At the following prompt, indicate whether you want to add more disks as spares (**y**) or return to the `vxdiskadm` main menu (**n**):

   ```
   Mark another disk as a spare? [y,n,q,?] (default: n)
   ```

 Any VM disk in this disk group can now use this disk as a spare in the event of a failure. If a disk fails, hot-relocation should automatically occur (if possible). You should be notified of the failure and relocation through

electronic mail. After successful relocation, you may want to replace the failed disk.

Removing a disk from use as a hot-relocation spare

While a disk is designated as a spare, the space on that disk is not used for the creation of VxVM objects within its disk group. If necessary, you can free a spare disk for general use by removing it from the pool of hot-relocation disks.

To remove a spare from the hot-relocation pool, use the following command:

```
# vxedit [-g diskgroup] set spare=off diskname
```

where *diskname* is the disk media name.

For example, to make `mydg01` available for normal use in the disk group, `mydg`, use the following command:

```
# vxedit -g mydg set spare=off mydg01
```

To use vxdiskadm to remove a disk from the hot-relocation pool

1. Select menu item `12 (Turn off the spare flag on a disk)` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name of a spare disk (such as `mydg01`):

```
Menu: VolumeManager/Disk/UnmarkSpareDisk

Use this operation to turn off the spare flag on a disk.
This operation takes, as input, a disk name. This is the same
name that you gave to the disk when you added the disk to the
disk group.

Enter disk name [<disk>,list,q,?] mydg01
```

 The following confirmation is displayed:

```
VxVM NOTICE V-5-2-143 Disk mydg01 in mydg no longer marked as
a spare disk.
```

3. At the following prompt, indicate whether you want to disable more spare disks (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Turn-off spare flag on another disk? [y,n,q,?] (default: n)
```

Excluding a disk from hot-relocation use

To exclude a disk from hot-relocation use, use the following command:

```
# vxedit [-g diskgroup] set nohotuse=on diskname
```

where *diskname* is the disk media name.

To use vxdiskadm to exclude a disk from hot-relocation use

1. Select menu item `15 (Exclude a disk from hot-relocation use)` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name (such as `mydg01`):

```
Exclude a disk from hot-relocation use
Menu: VolumeManager/Disk/UnmarkSpareDisk

Use this operation to exclude a disk from hot-relocation use.
This operation takes, as input, a disk name. This is the same
name that you gave to the disk when you added the disk to the
disk group.

Enter disk name [<disk>,list,q,?] mydg01
```

 The following confirmation is displayed:

```
VxVM INFO V-5-2-925 Excluding mydg01 in mydg from hot-
relocation use is complete.
```

3. At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Exclude another disk from hot-relocation use? [y,n,q,?]
(default: n)
```

Making a disk available for hot-relocation use

Free space is used automatically by hot-relocation in case spare space is not sufficient to relocate failed subdisks. You can limit this free space usage by hot-relocation by specifying which free disks should not be touched by hot-relocation. If a disk was previously excluded from hot-relocation use, you can undo the exclusion and add the disk back to the hot-relocation pool.

To make a disk available for hot-relocation use, use the following command:

```
# vxedit [-g diskgroup] set nohotuse=off diskname
```

To use vxdiskadm to make a disk available for hot-relocation use

1. Select menu item `16 (Make a disk available for hot-relocation use)` from the `vxdiskadm` main menu.
2. At the following prompt, enter the disk media name (such as `mydg01`):

```
Menu: VolumeManager/Disk/UnmarkSpareDisk

Use this operation to make a disk available for hot-relocation
use. This only applies to disks that were previously excluded
from hot-relocation use. This operation takes, as input, a
disk name. This is the same name that you gave to the disk when
you added the disk to the disk group.
```

```
Enter disk name [<disk>,list,q,?] mydg01
```

The following confirmation is displayed:

```
V-5-2-932 Making mydg01 in mydg available for hot-relocation
use is complete.
```

3 At the following prompt, indicate whether you want to add more disks to be excluded from hot-relocation (**y**) or return to the `vxdiskadm` main menu (**n**):

```
Make another disk available for hot-relocation use? [y,n,q,?]
(default: n)
```

Configuring hot-relocation to use only spare disks

If you want VxVM to use only spare disks for hot-relocation, add the following line to the file `/etc/default/vxassist`:

```
spare=only
```

If not enough storage can be located on disks marked as spare, the relocation fails. Any free space on non-spare disks is not used.

Moving and unrelocating subdisks

When hot-relocation occurs, subdisks are relocated to spare disks and/or available free space within the disk group. The new subdisk locations may not provide the same performance or data layout that existed before hot-relocation took place. You can move the relocated subdisks (after hot-relocation is complete) to improve performance.

You can also move the relocated subdisks off the spare disks to keep the spare disk space free for future hot-relocation needs. Another reason for moving subdisks is to recreate the configuration that existed before hot-relocation occurred.

During hot-relocation, one of the electronic mail messages sent to `root` is shown in the following example:

```
To: root
Subject: Volume Manager failures on host teal

Attempting to relocate subdisk mydg02-03 from plex home-02.
Dev_offset 0 length 1164 dm_name mydg02 da_name c0t5d0.
The available plex home-01 will be used to recover the data.
```

This message has information about the subdisk before relocation and can be used to decide where to move the subdisk after relocation.

Here is an example message that shows the new location for the relocated subdisk:

```
To: root
Subject: Attempting VxVM relocation on host teal
```

```
Volume home Subdisk mydg02-03 relocated to mydg05-01,
but not yet recovered.
```

Before you move any relocated subdisks, fix or replace the disk that failed (as described in "Removing and replacing disks" on page 112). Once this is done, you can move a relocated subdisk back to the original disk as described in the following sections.

Caution: During subdisk move operations, RAID-5 volumes are not redundant.

Moving and unrelocating subdisks using vxdiskadm

To move the hot-relocated subdisks back to the disk where they originally resided after the disk has been replaced following a failure

1. Select menu item `14 (Unrelocate subdisks back to a disk)` from the `vxdiskadm` main menu.
2. This option prompts for the original disk media name first.
 Enter the disk media name where the hot-relocated subdisks originally resided at the following prompt:
   ```
   Enter the original disk name [<disk>,list,q,?]
   ```
 If there are no hot-relocated subdisks in the system, `vxdiskadm` displays `Currently there are no hot-relocated disks`, and asks you to press Return to continue.
3. You are next asked if you want to move the subdisks to a destination disk other than the original disk.
   ```
   Unrelocate to a new disk [y,n,q,?] (default: n)
   ```
4. If moving subdisks to their original offsets is not possible, you can choose to unrelocate the subdisks forcibly to the specified disk, but not necessarily to the same offsets.
   ```
   Use -f option to unrelocate the subdisks if moving to the exact
   offset fails? [y,n,q,?] (default: n)
   ```
5. If you entered **`y`** at step 4 to unrelocate the subdisks forcibly, enter **`y`** or press Return at the following prompt to confirm the operation:
   ```
   Requested operation is to move all the subdisks which were
   hot-relocated from mydg10 back to mydg10 of disk group mydg.
   Continue with operation? [y,n,q,?] (default: y)
   ```
 A status message is displayed at the end of the operation.
   ```
   VxVM INFO V-5-2-954 Unrelocate to disk mydg10 is complete.
   ```

As an alternative to this procedure, use either the `vxassist` command or the `vxunreloc` command directly, as described in "Moving and unrelocating

subdisks using vxassist” on page 392 and “Moving and unrelocating subdisks using vxunreloc” on page 392.

Moving and unrelocating subdisks using vxassist

You can use the `vxassist` command to move and unrelocate subdisks. For example, to move the relocated subdisks on `mydg05` belonging to the volume `home` back to `mydg02`, enter the following command:

```
# vxassist -g mydg move home !mydg05 mydg02
```

Here, `!mydg05` specifies the current location of the subdisks, and `mydg02` specifies where the subdisks should be relocated.

If the volume is enabled, subdisks within detached or disabled plexes, and detached log or RAID-5 subdisks, are moved without recovery of data.

If the volume is not enabled, subdisks within STALE or OFFLINE plexes, and stale log or RAID-5 subdisks, are moved without recovery. If there are other subdisks within a non-enabled volume that require moving, the relocation fails.

For enabled subdisks in enabled plexes within an enabled volume, data is moved to the new location, without loss of either availability or redundancy of the volume.

Moving and unrelocating subdisks using vxunreloc

VxVM hot-relocation allows the system to automatically react to I/O failures on a redundant VxVM object at the subdisk level and then take necessary action to make the object available again. This mechanism detects I/O failures in a subdisk, relocates the subdisk, and recovers the plex associated with the subdisk. After the disk has been replaced, `vxunreloc` allows you to restore the system back to the configuration that existed before the disk failure. `vxunreloc` allows you to move the hot-relocated subdisks back onto a disk that was replaced due to a failure.

When `vxunreloc` is invoked, you must specify the disk media name where the hot-relocated subdisks originally resided. When `vxunreloc` moves the subdisks, it moves them to the original offsets. If you try to unrelocate to a disk that is smaller than the original disk that failed,`vxunreloc` does nothing except return an error.

vxunreloc provides an option to move the subdisks to a different disk from where they were originally relocated. It also provides an option to unrelocate subdisks to a different offset as long as the destination disk is large enough to accommodate all the subdisks.

If `vxunreloc` cannot replace the subdisks back to the same original offsets, a force option is available that allows you to move the subdisks to a specified disk

without using the original offsets. Refer to the vxunreloc(1M) manual page for more information.

The examples in the following sections demonstrate the use of vxunreloc.

Moving hot-relocated subdisks back to their original disk

Assume that mydg01 failed and all the subdisks were relocated. After mydg01 is replaced, vxunreloc can be used to move all the hot-relocated subdisks back to mydg01.

```
# vxunreloc -g mydg mydg01
```

Moving hot-relocated subdisks back to a different disk

The vxunreloc utility provides the -n option to move the subdisks to a different disk from where they were originally relocated.

Assume that mydg01 failed, and that all of the subdisks that resided on it were hot-relocated to other disks. vxunreloc provides an option to move the subdisks to a different disk from where they were originally relocated. After the disk is repaired, it is added back to the disk group using a different name, for example, mydg05. If you want to move all the hot-relocated subdisks back to the new disk, the following command can be used:

```
# vxunreloc -g mydg -n mydg05 mydg01
```

The destination disk should have at least as much storage capacity as was in use on the original disk. If there is not enough space, the unrelocate operation will fail and none of the subdisks will be moved.

Forcing hot-relocated subdisks to accept different offsets

By default, vxunreloc attempts to move hot-relocated subdisks to their original offsets. However, vxunreloc fails if any subdisks already occupy part or all of the area on the destination disk. In such a case, you have two choices:

- Move the existing subdisks somewhere else, and then re-run vxunreloc .
- Use the -f option provided by vxunreloc to move the subdisks to the destination disk, but leave it to vxunreloc to find the space on the disk. As long as the destination disk is large enough so that the region of the disk for storing subdisks can accommodate all subdisks, all the hot-relocated subdisks will be unrelocated without using the original offsets.

Assume that mydg01 failed and the subdisks were relocated and that you want to move the hot-relocated subdisks to mydg05 where some subdisks already reside. You can use the force option to move the hot-relocated subdisks to mydg05, but not to the exact offsets:

```
# vxunreloc -g mydg -f -n mydg05 mydg01
```

Examining which subdisks were hot-relocated from a disk

If a subdisk was hot relocated more than once due to multiple disk failures, it can still be unrelocated back to its original location. For instance, if `mydg01` failed and a subdisk named `mydg01-01` was moved to `mydg02`, and then `mydg02` experienced disk failure, all of the subdisks residing on it, including the one which was hot-relocated to it, will be moved again. When `mydg02` was replaced, a `vxunreloc` operation for `mydg02` will do nothing to the hot-relocated subdisk `mydg01-01`. However, a replacement of `mydg01` followed by a `vxunreloc` operation, moves `mydg01-01` back to `mydg01` if `vxunreloc` is run immediately after the replacement.

After the disk that experienced the failure is fixed or replaced, `vxunreloc` can be used to move all the hot-relocated subdisks back to the disk. When a subdisk is hot-relocated, its original disk-media name and the offset into the disk are saved in the configuration database. When a subdisk is moved back to the original disk or to a new disk using `vxunreloc`, the information is erased. The original disk-media name and the original offset are saved in the subdisk records. To print all of the subdisks that were hot-relocated from `mydg01` in the `mydg` disk group, use the following command:

```
# vxprint -g mydg -se 'sd_orig_dmname="mydg01"'
```

Restarting vxunreloc after errors

`vxunreloc` moves subdisks in three phases:

- `vxunreloc` creates as many subdisks on the specified destination disk as there are subdisks to be unrelocated. The string `UNRELOC` is placed in the `comment` field of each subdisk record.
 Creating the subdisk is an *all-or-nothing* operation. If `vxunreloc` cannot create all the subdisks successfully, none are created, and `vxunreloc` exits.
- `vxunreloc` moves the data from each subdisk to the corresponding newly created subdisk on the destination disk.
- When all subdisk data moves have been completed successfully, `vxunreloc` sets the `comment` field to the null string for each subdisk on the destination disk whose `comment` field is currently set to `UNRELOC`.

The `comment` fields of all the subdisks on the destination disk remain marked as `UNRELOC` until phase 3 completes. If its execution is interrupted, `vxunreloc` can subsequently re-use subdisks that it created on the destination disk during a previous execution, but it does not use any data that was moved to the destination disk.

If a subdisk data move fails, `vxunreloc` displays an error message and exits. Determine the problem that caused the move to fail, and fix it before re-executing `vxunreloc`.

If the system goes down after the new subdisks are created on the destination disk, but before all the data has been moved, re-execute `vxunreloc` when the system has been rebooted.

Caution: Do not modify the string `UNRELOC` in the comment field of a subdisk record.

Modifying the behavior of hot-relocation

Hot-relocation is turned on as long as the `vxrelocd` process is running. You should normally leave hot-relocation turned on so that you can take advantage of this feature if a failure occurs. However, if you choose to disable hot-relocation (perhaps because you do not want the free space on your disks to be used for relocation), you can prevent `vxrelocd` from starting at system startup time by editing the startup file that invokes `vxrelocd`: `/sbin/init.d/vxvm-recover`.

You can alter the behavior of `vxrelocd` as follows:

- To prevent `vxrelocd` starting, comment out the entry that invokes it in the startup file:

  ```
  # nohup vxrelocd root &
  ```

- By default, `vxrelocd` sends electronic mail to `root` when failures are detected and relocation actions are performed. You can instruct `vxrelocd` to notify additional users by adding the appropriate user names as shown here:

  ```
  nohup vxrelocd root user1 user2 &
  ```

- To reduce the impact of recovery on system performance, you can instruct `vxrelocd` to increase the delay between the recovery of each region of the volume, as shown in the following example:

  ```
  nohup vxrelocd -o slow[=IOdelay] root &
  ```

 where the optional *IOdelay* value indicates the desired delay in milliseconds. The default value for the delay is 250 milliseconds.

After making changes to the way `vxrelocd` is invoked in the startup file, reboot the system so that the changes go into effect.

You can also stop hot-relocation at any time by killing the `vxrelocd` process (this should not be done while a hot-relocation attempt is in progress).

When executing `vxrelocd` manually, either include `/etc/vx/bin` in your PATH or specify `vxrelocd`'s absolute pathname, for example:

```
# PATH=/etc/vx/bin:$PATH
# export PATH
# nohup vxrelocd root &
```

Alternatively, you can use the following command:

```
# nohup /etc/vx/bin/vxrelocd root user1 user2 &
```

See the `vxrelocd`(1M) manual page for more information.

Chapter 13

Administering cluster functionality

A cluster consists of a number of hosts or nodes that share a set of disks. The main benefits of cluster configurations are:

Availability	If one node fails, the other nodes can still access the shared disks. When configured with suitable software, mission-critical applications can continue running by transferring their execution to a standby node in the cluster. This ability to provide continuous uninterrupted service by switching to redundant hardware is commonly termed *failover*. Failover is transparent to users and high-level applications for database and file-sharing. You must configure cluster management software, such as Veritas Cluster Server (VCS), to monitor systems and services, and to restart applications on another node in the event of either hardware or software failure. VCS also allows you to perform general administration tasks such as making nodes join or leave a cluster. Note that a standby node need not remain idle. It could be used to serve other applications in parallel.
Off-host processing	Clusters can reduce contention for system resources by performing activities such as backup, decision support and report generation on the more lightly loaded nodes of the cluster. This allows businesses to derive enhanced value from their investment in cluster systems.

The cluster functionality of Veritas Volume Manager (CVM) allows up to 16 nodes in a cluster to simultaneously access and manage a set of disks under VxVM control (VM disks). The same logical view of disk configuration and any changes to this is available on all the nodes. When the cluster functionality is

enabled, all the nodes in the cluster can share VxVM objects such as shared disk groups. Private disk groups are supported in the same way as in a non-clustered environment. This chapter discusses the cluster functionality that is provided with VxVM.

Note: You need an additional license to use this feature.

This chapter does not discuss Veritas Storage Foundation Cluster File System (SFCFS) nor cluster management software such as Veritas Cluster Server (VCS). Such products are separately licensed, and are not included with Veritas Volume Manager. See the documentation provided with those products for more information about them.

For information about administering a cluster that is under the control of HP Serviceguard, refer to the HP Serviceguard Storage Management Suite documentation.

For additional information about using the Dynamic Multipathing (DMP) feature of VxVM in a clustered environment, see "DMP in a clustered environment" on page 132.

For information about administering campus cluster configurations (also known as stretch cluster or remote mirror configurations), see "Administering sites and remote mirrors" on page 431.

Overview of cluster volume management

In recent years, tightly-coupled cluster systems have become increasingly popular in the realm of enterprise-scale mission-critical data processing. The primary advantage of clusters is protection against hardware failure. Should the primary node fail or otherwise become unavailable, applications can continue to run by transferring their execution to standby nodes in the cluster. This ability to provide continuous availability of service by switching to redundant hardware is commonly termed *failover*.

Another major advantage of clustered systems is their ability to reduce contention for system resources caused by activities such as backup, decision support and report generation. Businesses can derive enhanced value from their investment in cluster systems by performing such operations on lightly loaded nodes in the cluster rather than on the heavily loaded nodes that answer requests for service. This ability to perform some operations on the lightly loaded nodes is commonly termed *load balancing*.

The cluster functionality of VxVM works together with the cluster monitor daemon that is provided by VCS or by the host operating system. When configured correctly, the cluster monitor informs VxVM of changes in cluster

membership. Each node starts up independently and has its own cluster monitor plus its own copies of the operating system and VxVM with support for cluster functionality. When a node joins a cluster, it gains access to shared disk groups and volumes. When a node leaves a cluster, it no longer has access to these shared objects. A node joins a cluster when you issue the appropriate command on that node. In an HP Serviceguard cluster, a node can join the cluster automatically at boot time.

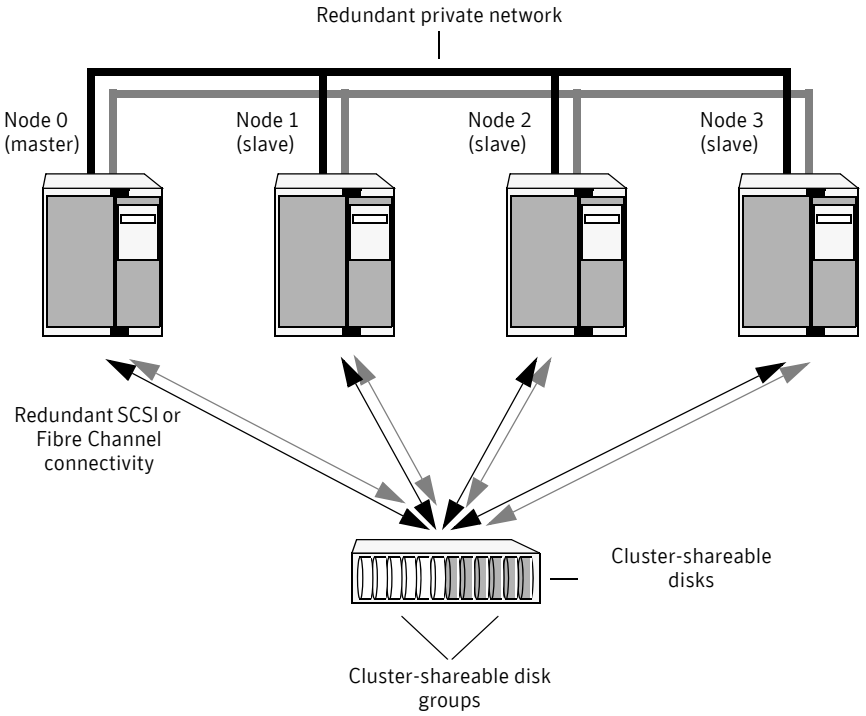
Caution: The cluster functionality of VxVM is supported only when used in conjunction with a cluster monitor that has been configured correctly to work with VxVM.

Figure 13-1 illustrates a simple cluster arrangement consisting of four nodes with similar or identical hardware characteristics (CPUs, RAM and host adapters), and configured with identical software (including the operating system). The nodes are fully connected by a private network and they are also separately connected to shared external storage (either disk arrays or JBODs: *just a bunch of disks*) via SCSI or via Fibre Channel in a Storage Area Network (SAN).

Note: In this example, each node has two independent paths to the disks, which are configured in one or more cluster-shareable disk groups. Multiple paths provide resilience against failure of one of the paths, but this is not a requirement for cluster configuration. Disks may also be connected by single paths.

The private network allows the nodes to share information about system resources and about each other's state. Using the private network, any node can recognize which other nodes are currently active, which are joining or leaving the cluster, and which have failed. The private network requires at least two communication channels to provide redundancy against one of the channels failing. If only one channel were used, its failure would be indistinguishable from node failure—a condition known as *network partitioning*.

Figure 13-1 Example of a 4-node cluster



To the cluster monitor, all nodes are the same. VxVM objects configured within shared disk groups can potentially be accessed by all nodes that join the cluster. However, the cluster functionality of VxVM requires that one node act as the master node; all other nodes in the cluster are slave nodes. Any node is capable of being the master node, and it is responsible for coordinating certain VxVM activities.

Note: You must run commands that configure or reconfigure VxVM objects on the master node. Tasks that must be initiated from the master node include setting up shared disk groups, creating and reconfiguring volumes, and performing snapshot operations.

VxVM determines that the first node to join a cluster performs the function of master node. If the master node leaves a cluster, one of the slave nodes is chosen to be the new master. In "Example of a 4-node cluster," node 0 is the master node and nodes 1, 2 and 3 are slave nodes.

Private and shared disk groups

Two types of disk groups are defined:

Private disk group	Belongs to only one node. A private disk group can only be imported by one system at a time. Disks in a private disk group may be physically accessible from one or more systems, but access is restricted to one system only. The boot disk group (usually aliased by the reserved disk group name `bootdg`) is always a private disk group.
Shared disk group	Can be shared by all nodes. A shared (or cluster-shareable) disk group is imported by all cluster nodes. Disks in a shared disk group must be physically accessible from all systems that may join the cluster.

In a cluster, most disk groups are shared. Disks in a shared disk group are accessible from all nodes in a cluster, allowing applications on multiple cluster nodes to simultaneously access the same disk. A volume in a shared disk group can be simultaneously accessed by more than one node in the cluster, subject to licensing and disk group activation mode restrictions.

You can use the `vxdg` command to designate a disk group as cluster-shareable as described in "Importing disk groups as shared" on page 423. When a disk group is imported as cluster-shareable for one node, each disk header is marked with the cluster ID. As each node subsequently joins the cluster, it recognizes the disk group as being cluster-shareable and imports it. As system administrator, you can also import or deport a shared disk group at any time; the operation takes place in a distributed fashion on all nodes.

Each physical disk is marked with a unique disk ID. When cluster functionality for VxVM starts on the master, it imports all shared disk groups (except for any that have the `noautoimport` attribute set). When a slave tries to join a cluster, the master sends it a list of the disk IDs that it has imported, and the slave checks to see if it can access them all. If the slave cannot access one of the listed disks, it abandons its attempt to join the cluster. If it can access all of the listed disks, it joins the cluster and imports the same shared disk groups as the master. When a node leaves the cluster gracefully, it deports all its imported shared disk groups, but they remain imported on the surviving nodes.

Reconfiguring a shared disk group is performed with the cooperation of all nodes. Configuration changes to the disk group happen simultaneously on all nodes and the changes are identical. Such changes are *atomic* in nature, which means that they either occur simultaneously on all nodes or not at all.

Whether all members of the cluster have simultaneous read and write access to a cluster-shareable disk group depends on its activation mode setting as discussed in "Activation modes of shared disk groups." The data contained in a

cluster-shareable disk group is available as long as at least one node is active in the cluster. The failure of a cluster node does not affect access by the remaining active nodes. Regardless of which node accesses a cluster-shareable disk group, the configuration of the disk group looks the same.

Note: Applications running on each node can access the data on the VM disks simultaneously. VxVM does not protect against simultaneous writes to shared volumes by more than one node. It is assumed that applications control consistency (by using a distributed lock manager, for example).

Activation modes of shared disk groups

A shared disk group must be activated on a node in order for the volumes in the disk group to become accessible for application I/O from that node. The ability of applications to read from or to write to volumes is dictated by the activation mode of a shared disk group. Valid activation modes for a shared disk group are `exclusivewrite`, `readonly`, `sharedread`, `sharedwrite`, and `off` (inactive). These activation modes are described in detail in the table "Activation modes for shared disk groups."

Note: The default activation mode for shared disk groups is `off` (inactive).

Special uses of clusters, such as high availability (HA) applications and off-host backup, can use disk group activation to explicitly control volume access from different nodes in the cluster.

Table 13-1 Activation modes for shared disk groups

Activation mode	Description
`exclusivewrite (ew)`	The node has exclusive write access to the disk group. No other node can activate the disk group for write access.
`readonly (ro)`	The node has read access to the disk group and denies write access for all other nodes in the cluster. The node has no write access to the disk group. Attempts to activate a disk group for either of the write modes on other nodes fail.
`sharedread (sr)`	The node has read access to the disk group. The node has no write access to the disk group, however other nodes can obtain write access.

Table 13-1 Activation modes for shared disk groups

Activation mode	Description
`sharedwrite (sw)`	The node has write access to the disk group. Attempts to activate the disk group for shared read and shared write access succeed. Attempts to activate the disk group for exclusive write and read-only access fail.
`off`	The node has neither read nor write access to the disk group. Query operations on the disk group are permitted.

The following table summarizes the allowed and conflicting activation modes for shared disk groups:

Table 13-2 Allowed and conflicting activation modes

Disk group activated in cluster as...	Attempt to activate disk group on another node as...			
	exclusive-write	readonly	sharedread	sharedwrite
exclusivewrite	Fails	Fails	Succeeds	Fails
readonly	Fails	Succeeds	Succeeds	Fails
sharedread	Succeeds	Succeeds	Succeeds	Succeeds
sharedwrite	Fails	Fails	Succeeds	Succeeds

Shared disk groups can be automatically activated in any mode during disk group creation or during manual or auto-import. To control auto-activation of shared disk groups, the defaults file `/etc/default/vxdg` must be created.

The defaults file `/etc/default/vxdg` must contain the following lines:

```
enable_activation=true
default_activation_mode=activation-mode
```

The *activation-mode* is one of `exclusivewrite`, `readonly`, `sharedread`, `sharedwrite`, or `off`.

When a shared disk group is created or imported, it is activated in the specified mode. When a node joins the cluster, all shared disk groups accessible from the node are activated in the specified mode.

Note: The activation mode of a disk group controls volume I/O from different nodes in the cluster. It is not possible to activate a disk group on a given node if it is activated in a conflicting mode on another node in the cluster. When enabling activation using the defaults file, it is recommended that this file be made identical on all nodes in the cluster. Otherwise, the results of activation are unpredictable.

If the defaults file is edited while the `vxconfigd` daemon is already running, run the `vxconfigd -k` command on all nodes to restart the process.

If the default activation mode is anything other than `off`, an activation following a cluster join, or a disk group creation or import can fail if another node in the cluster has activated the disk group in a conflicting mode.

To display the activation mode for a shared disk group, use the `vxdg list` *diskgroup* command as described in "Listing shared disk groups" on page 421.

You can also use the `vxdg` command to change the activation mode on a shared disk group as described in "Changing the activation mode on a shared disk group" on page 425.

For a description of how to configure a volume so that it can only be opened by a single node in a cluster, see "Creating volumes with exclusive open access by a node" on page 426 and "Setting exclusive open access to a volume by a node" on page 426.

Connectivity policy of shared disk groups

A shared disk group provides concurrent read and write access to the volumes that it contains for all nodes in a cluster. A shared disk group can only be created on the master node. This has the following advantages and implications:

- All nodes in the cluster see exactly the same configuration.
- Only the master node can change the configuration.
- Any changes on the master node are automatically coordinated and propagated to the slave nodes in the cluster.
- Any failures that require a configuration change must be sent to the master node so that they can be resolved correctly.
- As the master node resolves failures, all the slave nodes are correctly updated. This ensures that all nodes have the same view of the configuration.

The practical implication of this design is that I/O failure on any node results in the configuration of all nodes being changed. This is known as the *global detach*

policy. However, in some cases, it is not desirable to have all nodes react in this way to I/O failure. To address this, an alternate way of responding to I/O failures, known as the *local detach policy*, was introduced in release 3.2 of VxVM.

The local detach policy is intended for use with shared mirrored volumes in a cluster. This policy prevents I/O failure on a single slave node from causing a plex to be detached. This would require the plex to be resynchronized when it is subsequently reattached. The local detach policy is available for disk groups that have a version number of 70 or greater.

Note: For small mirrored volumes, non-mirrored volumes, volumes that use hardware mirrors, and volumes in private disk groups, there is no benefit in configuring the local detach policy. In most cases, it is recommended that you use the default global detach policy.

The detach policies have no effect if the master node loses access to all copies of the configuration database and logs in a disk group. If this happened in releases prior to 4.1, the master node always disabled the disk group. Release 4.1 introduces the *disk group failure policy*, which allows you to change this behavior for critical disk groups. This policy is only available for disk groups that have a version number of 120 or greater.

The following sections describe the detach and failure policies in greater detail.

Global detach policy

Caution: The global detach policy must be selected when Dynamic MultiPathing (DMP) is used to manage multipathing on Active/Passive arrays, This ensures that all nodes correctly coordinate their use of the active path.

The global detach policy is the traditional and default policy for all nodes on the configuration. If there is a read or write I/O failure on a slave node, the master node performs the usual I/O recovery operations to repair the failure, and, if required, the plex is detached cluster-wide. All nodes remain in the cluster and continue to perform I/O, but the redundancy of the mirrors is reduced. When the problem that caused the I/O failure has been corrected, the mirrors that were detached must be recovered before the redundancy of the data can be restored.

Local detach policy

Caution: Do not use the local detach policy if you use the VCS agents that monitor the cluster functionality of Veritas Volume Manager, and which are provided with Veritas Storage Foundation™ for Cluster File System HA and Veritas Storage Foundation for databases HA. These agents do not notify VCS about local failures.

The local detach policy is designed to support failover applications in large clusters where the redundancy of the volume is more important than the number of nodes that can access the volume. If there is a write failure on a slave node, the master node performs the usual I/O recovery operations to repair the failure, and additionally contacts all the nodes to see if the disk is still acceptable to them. If the write failure is not seen by all the nodes, I/O is stopped for the node that first saw the failure, and the application using the volume is also notified about the failure. The volume is not disabled.

If required, configure the cluster management software to move the application to a different node, and/or remove the node that saw the failure from the cluster. The volume continues to return write errors, as long as one mirror of the volume has an error. The volume continues to satisfy read requests as long as one good plex is available.

If the reason for the I/O error is corrected and the node is still a member of the cluster, it can resume performing I/O from/to the volume without affecting the redundancy of the data.

See "Setting the disk detach policy on a shared disk group" on page 425 for information on how to use the `vxdg` command to set the disk detach policy on a shared disk group.

The table, "Cluster behavior under I/O failure to a mirrored volume for different disk detach policies," summarizes the effect on a cluster of I/O failure to the disks in a mirrored volume:

Table 13-3 Cluster behavior under I/O failure to a mirrored volume for different disk detach policies

Type of I/O failure	Local (diskdetpolicy=local)	Global (diskdetpolicy=global)
Failure of path to one disk in a volume for a single node	Reads fail only if no plexes remain available to the affected node. Writes to the volume fail.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.

Table 13-3 Cluster behavior under I/O failure to a mirrored volume for different disk detach policies

Type of I/O failure	Local (diskdetpolicy=local)	Global (diskdetpolicy=global)
Failure of paths to all disks in a volume for a single node	I/O fails for the affected node.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.
Failure of one or more disks in a volume for all nodes.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.	The plex is detached, and I/O from/to the volume continues. An I/O error is generated if no plexes remain.

Disk group failure policy

The local detach policy by itself is insufficient to determine the desired behavior if the master node loses access to all disks that contain copies of the configuration database and logs. In this case, the disk group is disabled. As a result, the other nodes in the cluster also lose access to the volume. In release 4.1, the disk group failure policy is introduced to determine the behavior of the master node in such cases. This policy has two possible settings as shown in the following table:

Table 13-4 Behavior of master node for different failure policies

Type of I/O failure	Leave (dgfailpolicy=leave)	Disable (dgfailpolicy=dgdisable)
Master node loses access to all copies of the logs.	The master node panics with the message “klog update failed” for a failed kernel-initiated transaction, or “cvm config update failed” for a failed user-initiated transaction.	The master node disables the disk group.

The behavior of the master node under the disk group failure policy is independent of the setting of the disk detach policy. If the disk group failure policy is set to `leave`, all nodes panic in the unlikely case that none of them can access the log copies.

See “Setting the disk group failure policy on a shared disk group” on page 426 for information on how to use the `vxdg` command to set the failure policy on a shared disk group.

Guidelines for choosing detach and failure policies

In most cases it is recommended that you use the global detach policy, and particularly if any of the following conditions apply:

- If you are using the VCS agents that monitor the cluster functionality of Veritas Volume Manager, and which are provided with Veritas Storage Foundation™ for Cluster File System HA and Veritas Storage Foundation for databases HA. These agents do not notify VCS about local failures.
- When an array is seen by DMP as Active/Passive. The local detach policy causes unpredictable behavior for Active/Passive arrays.
- For clusters with four or fewer nodes. With a small number of nodes in a cluster, it is preferable to keep all nodes actively using the volumes, and to keep the applications running on all the nodes, rather than keep the redundancy of the volume at the same level.
- If only non-mirrored, small mirrored, or hardware mirrored volumes are configured. This avoids the system overhead of the extra messaging that is required by the local detach policy.

The local detach policy may be suitable in the following cases:

- When large mirrored volumes are configured. Resynchronizing a reattached plex can degrade system performance. The local detach policy can avoid the need to detach the plex at all. (Alternatively, the dirty region logging (DRL) feature can be used to reduce the amount of resynchronization that is required.)
- For clusters with more than four nodes. Keeping an application running on a particular node is less critical when there are many nodes in a cluster. It may be possible to configure the cluster management software to move an application to a node that has access to the volumes. In addition, load balancing may be able to move applications to a different volume from the one that experienced the I/O problem. This preserves data redundancy, and other nodes may still be able to perform I/O from/to the volumes on the disk.

If you have a critical disk group that you do not want to become disabled in the case that the master node loses access to the copies of the logs, set the disk group failure policy to `leave`. This prevents I/O failure on the master node disabling the disk group. However, critical applications running on the master node fail if they lose access to the other shared disk groups. In such a case, it may be preferable to set the policy to `dgdisable`, and to allow the disk group to be disabled.

The default settings for the detach and failure policies are `global` and `dgdisable` respectively. You can use the `vxdg` command to change both the detach and failure policies on a shared disk group, as shown in this example:

```
# vxdg -g diskgroup set diskdetpolicy=local dgfailpolicy=leave
```

Effect of disk connectivity on cluster reconfiguration

The detach policy, previous I/O errors, or access to disks are not considered when a new master node is chosen. When the master node leaves a cluster, the node that takes over as master of the cluster may already have seen I/O failures for one or more disks. Under the local detach policy, if a node was affected before reconfiguration, and this node then becomes the master, the failure is treated as described in "Connectivity policy of shared disk groups" on page 404. Some failure scenarios do not result in a disk group failure policy being invoked, but can potentially impact the cluster. For example, if the local disk detach policy is in effect, and the new master node has a failed plex, this results in all nodes detaching the plex because the new master is unaffected by the policy.

The detach policy does not change the requirement that a node joining a cluster must have access to all the disks in all shared disk groups. Similarly, a node that is removed from the cluster because of an I/O failure cannot rejoin the cluster until this requirement is met.

Limitations of shared disk groups

Note: The boot disk group (usually aliased as `bootdg`) cannot be made cluster-shareable. It must be private.

Only raw device access may be performed via the cluster functionality of VxVM. It does not support shared access to file systems in shared volumes unless the appropriate software is installed and configured.

The cluster functionality of VxVM does not support RAID-5 volumes, or task monitoring for cluster-shareable disk groups. These features can, however, be used in private disk groups that are attached to specific nodes of a cluster.

If you have RAID-5 volumes in a private disk group that you wish to make shareable, you must first relayout the volumes as a supported volume type such as `stripe-mirror` or `mirror-stripe`. Online relayout of shared volumes is supported provided that it does not involve RAID-5 volumes.

If a shared disk group contains RAID-5 volumes, deport it and then reimport the disk group as private on one of the cluster nodes. Reorganize the volumes into layouts that are supported for shared disk groups, and then deport and reimport the disk group as shared.

Cluster initialization and configuration

Before any nodes can join a new cluster for the first time, you must supply certain configuration information during cluster monitor setup. This information is normally stored in some form of cluster monitor configuration database. The precise content and format of this information depends on the characteristics of the cluster monitor. The information required by VxVM is as follows:

- cluster ID
- node IDs
- network addresses of nodes
- port addresses

When a node joins the cluster, this information is automatically loaded into VxVM on that node at node startup time.

Note: To make effective use of the cluster functionality of VxVM requires that you configure a cluster monitor (such as provided by HP Serviceguard, or by GAB (Group Membership and Atomic Broadcast) in VCS). If HP Serviceguard is used as the cluster monitor, no additional configuration of VxVM is required, apart from the cluster configuration requirements of HP Serviceguard.

The cluster monitor startup procedure effects node initialization, and brings up the various cluster components (such as VxVM with cluster support, the cluster monitor, and a distributed lock manager) on the node. Once this is complete, applications may be started. The cluster monitor startup procedure must be invoked on each node to be joined to the cluster.

For VxVM in a cluster environment, initialization consists of loading the cluster configuration information and joining the nodes in the cluster. The first node to join becomes the master node, and later nodes (slaves) join to the master. If two nodes join simultaneously, VxVM chooses the master. Once the join for a given node is complete, that node has access to the shared disk groups and volumes.

Cluster reconfiguration

Cluster reconfiguration occurs if a node leaves or joins a cluster. Each node's cluster monitor continuously watches the other cluster nodes. When the membership of the cluster changes, the cluster monitor informs VxVM for it to take appropriate action.

During cluster reconfiguration, VxVM suspends I/O to shared disks. I/O resumes when the reconfiguration completes. Applications may appear to freeze for a short time during reconfiguration.

If other operations, such as VxVM operations or recoveries, are in progress, cluster reconfiguration can be delayed until those operations have completed. Volume reconfigurations (see "Volume reconfiguration" on page 413) do not take place at the same time as cluster reconfigurations. Depending on the circumstances, an operation may be held up and restarted later. In most cases, cluster reconfiguration takes precedence. However, if the volume reconfiguration is in the commit stage, it completes first.

For more information on cluster reconfiguration, see "vxclustadm utility" on page 411.

vxclustadm utility

The `vxclustadm` command provides an interface to the cluster functionality of VxVM when VCS or HP Serviceguard is used as the cluster monitor. It is also called during cluster startup and shutdown.

The `startnode` keyword to `vxclustadm` starts cluster functionality on a cluster node by passing cluster configuration information to the VxVM kernel. In response to this command, the kernel and the VxVM configuration daemon, `vxconfigd`, perform initialization.

The `stopnode` keyword stops cluster functionality on a node. It waits for all outstanding I/O to complete and for all applications to close shared volumes.

The `abortnode` keyword terminates cluster activity on a node. It does not wait for outstanding I/O to complete nor for applications to close shared volumes.

The `reinit` keyword allows nodes to be added to or removed from a cluster without stopping the cluster. Before running this command, the cluster configuration file must have been updated with information about the supported nodes in the cluster.

The `nidmap` keyword prints a table showing the mapping between CVM node IDs in VxVM's cluster-support subsystem and node IDs in the cluster monitor. It also prints the state of the node in the cluster.

The `nodestate` keyword reports the state of a cluster node and also the reason for the last abort of the node as shown in this example:

```
# /etc/vx/bin/vxclustadm nodestate
state: out of cluster
reason: user initiated stop
```

The various reasons that may be given are shown in Table 13-5.

Table 13-5 Node abort messages

Reason	Description
`cannot find disk on slave node`	Missing disk or bad disk on the slave node.
`cannot obtain configuration data`	The node cannot read the configuration data due to an error such as disk failure.
`cluster device open failed`	Open of a cluster device failed.
`clustering license mismatch with master node`	Clustering license does not match that on the master node.
`clustering license not available`	Clustering license cannot be found.
`connection refused by master`	Join of a node refused by the master node.
`disk in use by another cluster`	A disk belongs to a cluster other than the one that a node is joining.
`join timed out during reconfiguration`	Join of a node has timed out due to reconfiguration taking place in the cluster.
`klog update failed`	Cannot update kernel log copies during the join of a node.
`master aborted during join`	Master node aborted while another node was joining the cluster.
`minor number conflict`	Minor number conflicts exist between private disk groups and shared disk groups that are being imported.
`protocol version out of range`	Cluster protocol version mismatch or unsupported version.
`recovery in progress`	Volumes that were opened by the node are still recovering.
`transition to role failed`	Changing the role of a node to be the master failed.
`user initiated abort`	Node is out of cluster due to an abort initiated by the user or by the cluster monitor.
`user initiated stop`	Node is out of cluster due to a stop initiated by the user or by the cluster monitor.
`vxconfigd is not enabled`	The VxVM configuration daemon is not enabled.

See the vxclustadm(1M) manual page for more information about vxclustadm and for examples of its usage.

Volume reconfiguration

Volume reconfiguration is the process of creating, changing, and removing VxVM objects such as disk groups, volumes and plexes. In a cluster, all nodes co-operate to perform such operations. The vxconfigd daemons (see "vxconfigd daemon" on page 414) play an active role in volume reconfiguration. For reconfiguration to succeed, a vxconfigd daemon must be running on each of the nodes.

A volume reconfiguration *transaction* is initiated by running a VxVM utility on the master node. The utility contacts the local vxconfigd daemon on the master node, which validates the requested change. For example, vxconfigd rejects an attempt to create a new disk group with the same name as an existing disk group. The vxconfigd daemon on the master node then sends details of the changes to the vxconfigd daemons on the slave nodes. The vxconfigd daemons on the slave nodes then perform their own checking. For example, each slave node checks that it does not have a private disk group with the same name as the one being created; if the operation involves a new disk, each node checks that it can access that disk. When the vxconfigd daemons on all the nodes agree that the proposed change is reasonable, each notifies its kernel. The kernels then co-operate to either commit or to abandon the transaction. Before the transaction can be committed, all of the kernels ensure that no I/O is underway, and block any I/O issued by applications until the reconfiguration is complete. The master node is responsible both for initiating the reconfiguration, and for coordinating the commitment of the transaction. The resulting configuration changes appear to occur simultaneously on all nodes.

If a vxconfigd daemon on any node goes away during reconfiguration, all nodes are notified and the operation fails. If any node leaves the cluster, the operation fails unless the master has already committed it. If the master node leaves the cluster, the new master node, which was previously a slave node, completes or fails the operation depending on whether or not it received notification of successful completion from the previous master node. This notification is performed in such a way that if the new master does not receive it, neither does any other slave.

If a node attempts to join a cluster while a volume reconfiguration is being performed, the result of the reconfiguration depends on how far it has progressed. If the kernel has not yet been invoked, the volume reconfiguration is suspended until the node has joined the cluster. If the kernel has been invoked, the node waits until the reconfiguration is complete before joining the cluster.

When an error occurs, such as when a check on a slave fails or a node leaves the cluster, the error is returned to the utility and a message is sent to the console on the master node to identify on which node the error occurred.

vxconfigd daemon

The VxVM configuration daemon, `vxconfigd`, maintains the configuration of VxVM objects. It receives cluster-related instructions from the kernel. A separate copy of `vxconfigd` runs on each node, and these copies communicate with each other over a network. When invoked, a VxVM utility communicates with the `vxconfigd` daemon running on the same node; it does not attempt to connect with `vxconfigd` daemons on other nodes. During cluster startup, the kernel prompts `vxconfigd` to begin cluster operation and indicates whether it is a master node or a slave node.

When a node is initialized for cluster operation, the `vxconfigd` daemon is notified that the node is about to join the cluster and is provided with the following information from the cluster monitor configuration database:

- cluster ID
- node IDs
- master node ID
- role of the node
- network address of the `vxconfigd` daemon on each node (if applicable)

On the master node, the `vxconfigd` daemon sets up the shared configuration by importing shared disk groups, and informs the kernel when it is ready for the slave nodes to join the cluster.

On slave nodes, the `vxconfigd` daemon is notified when the slave node can join the cluster. When the slave node joins the cluster, the `vxconfigd` daemon and the VxVM kernel communicate with their counterparts on the master node to set up the shared configuration.

When a node leaves the cluster, the kernel notifies the `vxconfigd` daemon on all the other nodes. The master node then performs any necessary cleanup. If the master node leaves the cluster, the kernels select a new master node and the `vxconfigd` daemons on all nodes are notified of the choice.

The `vxconfigd` daemon also participates in volume reconfiguration as described in "Volume reconfiguration" on page 413.

vxconfigd daemon recovery

In a cluster, the `vxconfigd` daemons on the slave nodes are always connected to the `vxconfigd` daemon on the master node. If the `vxconfigd` daemon is

stopped, volume reconfiguration cannot take place. Other nodes can join the cluster if the vxconfigd daemon is not running on the slave nodes.

If the vxconfigd daemon stops, different actions are taken depending on which node this occurred:

- If the vxconfigd daemon is stopped on the master node, the vxconfigd daemons on the slave nodes periodically attempt to rejoin to the master node. Such attempts do not succeed until the vxconfigd daemon is restarted on the master. In this case, the vxconfigd daemons on the slave nodes have not lost information about the shared configuration, so that any displayed configuration information is correct.
- If the vxconfigd daemon is stopped on a slave node, the master node takes no action. When the vxconfigd daemon is restarted on the slave, the slave vxconfigd daemon attempts to reconnect to the master daemon and to re-acquire the information about the shared configuration. (Neither the kernel view of the shared configuration nor access to shared disks is affected.) Until the vxconfigd daemon on the slave node has successfully reconnected to the vxconfigd daemon on the master node, it has very little information about the shared configuration and any attempts to display or modify the shared configuration can fail. For example, shared disk groups listed using the vxdg list command are marked as disabled; when the rejoin completes successfully, they are marked as enabled.
- If the vxconfigd daemon is stopped on both the master and slave nodes, the slave nodes do not display accurate configuration information until vxconfigd is restarted on the master and slave nodes, and the daemons have reconnected.

If the CVM agent for VCS determines that the vxconfigd daemon is not running on a node during a cluster reconfiguration, vxconfigd is restarted automatically.

If it is necessary to restart vxconfigd manually in a VCS controlled cluster to resolve a VxVM issue, use this procedure:

1. Use the following command to disable failover on any service groups that contain VxVM objects:

   ```
   # hagrp -freeze group
   ```

2. Enter the following command to stop and restart the VxVM configuration daemon on the affected node:

   ```
   # vxconfigd -k
   ```

3. Use the following command to re-enable failover for the service groups that you froze in step 1:

   ```
   # hagrp -unfreeze group
   ```

Note: The `-r` reset option to `vxconfigd` restarts the `vxconfigd` daemon and recreates all states from scratch. This option cannot be used to restart `vxconfigd` while a node is joined to a cluster because it causes cluster information to be discarded.

In an HP Serviceguard cluster, use the equivalent Serviceguard functionality to stop and restart the appropriate package.

Node shutdown

Although it is possible to shut down the cluster on a node by invoking the shutdown procedure of the node's cluster monitor, this procedure is intended for terminating cluster components after stopping any applications on the node that have access to shared storage. VxVM supports *clean node shutdown*, which allows a node to leave the cluster gracefully when all access to shared volumes has ceased. The host is still operational, but cluster applications cannot be run on it.

The cluster functionality of VxVM maintains global state information for each volume. This enables VxVM to determine which volumes need to be recovered when a node crashes. When a node leaves the cluster due to a crash or by some other means that is not clean, VxVM determines which volumes may have writes that have not completed and the master node resynchronizes these volumes. It can use dirty region logging (DRL) or FastResync if these are active for any of the volumes.

Clean node shutdown must be used after, or in conjunction with, a procedure to halt all cluster applications. Depending on the characteristics of the clustered application and its shutdown procedure, a successful shutdown can require a lot of time (minutes to hours). For instance, many applications have the concept of *draining*, where they accept no new work, but complete any work in progress before exiting. This process can take a long time if, for example, a long-running transaction is active.

When the VxVM shutdown procedure is invoked, it checks all volumes in all shared disk groups on the node that is being shut down. The procedure then either continues with the shutdown, or fails for one of the following reasons:

- If all volumes in shared disk groups are closed, VxVM makes them unavailable to applications. Because all nodes are informed that these volumes are closed on the leaving node, no resynchronization is performed.
- If any volume in a shared disk group is open, the shutdown operation in the kernel waits until the volume is closed. There is no timeout checking in this operation.

Note: Once shutdown succeeds, the node has left the cluster. It is not possible to access the shared volumes until the node joins the cluster again.

Since shutdown can be a lengthy process, other reconfiguration can take place while shutdown is in progress. Normally, the shutdown attempt is suspended until the other reconfiguration completes. However, if it is already too far advanced, the shutdown may complete first.

Node abort

If a node does not leave a cluster cleanly, this is because it crashed or because some cluster component made the node leave on an emergency basis. The ensuing cluster reconfiguration calls the VxVM abort function. This procedure immediately attempts to halt all access to shared volumes, although it does wait until pending I/O from or to the disk completes.

I/O operations that have not yet been started are failed, and the shared volumes are removed. Applications that were accessing the shared volumes therefore fail with errors.

After a node abort or crash, shared volumes must be recovered, either by a surviving node or by a subsequent cluster restart, because it is very likely that there are unsynchronized mirrors.

Cluster shutdown

If all nodes leave a cluster, shared volumes must be recovered when the cluster is next started if the last node did not leave cleanly, or if resynchronization from previous nodes leaving uncleanly is incomplete.

Multiple host failover configurations

Outside the context of clustering functionality, VxVM disk groups can be “imported” (made available) from only one host at any given time. When a host imports a disk group as private, the volumes and configuration of that disk group become accessible to the host. If the administrator or system software wants to privately use the same disk group from another host, the host that already has the disk group imported (*importing host*) must “deport” (give up access to) the disk group. Once deported, the disk group can be imported by another host.

If two hosts are allowed to access a disk group concurrently without proper synchronization, such as that provided by the Oracle Parallel Server, the configuration of the disk group, and possibly the contents of volumes, can be

corrupted. Similar corruption can also occur if a file system or database on a raw disk partition is accessed concurrently by two hosts, so this problem in not limited to Veritas Volume Manager.

Import lock

When a host in a non-clustered environment imports a disk group, an import lock is written on all disks in that disk group. The import lock is cleared when the host deports the disk group. The presence of the import lock prevents other hosts from importing the disk group until the importing host has deported the disk group.

Specifically, when a host imports a disk group, the import normally fails if any disks within the disk group appear to be locked by another host. This allows automatic re-importing of disk groups after a reboot (*autoimporting*) and prevents imports by another host, even while the first host is shut down. If the importing host is shut down without deporting the disk group, the disk group can only be imported by another host by clearing the host ID lock first (discussed later).

The import lock contains a host ID (in Veritas Volume Manager, this is the host name) reference to identify the importing host and enforce the lock. Problems can therefore arise if two hosts have the same host ID.

Note: Since Veritas Volume Manager uses the host name as the host ID (by default), it is advisable to change the host name of one machine if another machine shares its host name. To change the host name, use the `vxdctl hostid` *new_hostname* command.

Failover

The import locking scheme works well in an environment where disk groups are not normally shifted from one system to another. However, consider a setup where two hosts, Node A and Node B, can access the drives of a disk group. The disk group is first imported by Node A, but the administrator wants to access the disk group from Node B if Node A crashes. This kind of scenario (*failover*) can be used to provide manual high availability to data, where the failure of one node does not prevent access to data. Failover can be combined with a “high availability” monitor to provide automatic high availability to data: when Node B detects that Node A has crashed or shut down, Node B imports (fails over) the disk group to provide access to the volumes.

Veritas Volume Manager can support failover, but it relies on the administrator or on an external high-availability monitor to ensure that the first system is shut down or unavailable before the disk group is imported to another system.

For details on how to clear locks and force an import, see "Moving disk groups between systems" on page 185 and the vxdg(1M) manual page.

Corruption of disk group configuration

If vxdg import is used with -C (clears locks) and/or -f (forces import) to import a disk group that is still in use from another host, disk group configuration corruption is likely to occur. Volume content corruption is also likely if a file system or database is started on the imported volumes before the other host crashes or shuts down.

If this kind of corruption occurs, you must probably rebuild your configuration from scratch and reload all volumes in the disk group from a backup. To backup and rebuild the configuration, if nothing has changed, use vxprint -mspvd and store the output which can be fed to vxmake to restore the layouts. There are typically numerous configuration copies for each disk group, but corruption nearly always affects all configuration copies, so redundancy does not help in this case.

Disk group configuration corruption usually shows up as missing or duplicate records in the configuration databases. This can result in a variety of vxconfigd error messages

VxVM vxconfigd ERROR V-5-1-569 Disk group *group*,Disk *disk*:Cannot auto-import group: *reason*

where the *reason* can describe errors such as:

```
Association not resolved
Association count is incorrect
Duplicate record in configuration
Configuration records are inconsistent
```

These errors are typically reported in association with specific disk group configuration copies, but usually apply to all copies. The following is usually displayed along with the error:

```
Disk group has no valid configuration copies
```

See the *Veritas Volume Manager Troubleshooting Guide* for more information on Veritas Volume Manager error messages.

If you use the Veritas Cluster Server product, all disk group failover issues can be managed correctly. VCS includes a high availability monitor and includes failover scripts for VxVM, VxFS, and for several popular databases.

The -t option to vxdg prevents automatic re-imports on reboot and is necessary when used with a host monitor (such as VCS) that controls imports itself, rather than relying on automatic imports by Veritas Volume Manager.

Administering VxVM in cluster environments

The following sections describe the administration of VxVM's cluster functionality.

Note: Most VxVM commands require superuser or equivalent privileges.

Requesting node status and discovering the master node

The `vxdctl` utility controls the operation of the `vxconfigd` volume configuration daemon. The `-c` option can be used to request cluster information and to find out which node is the master. To determine whether the `vxconfigd` daemon is enabled and/or running, use the following command:

```
# vxdctl -c mode
```

This produces different output messages depending on the current status of the cluster node:

Table 13-6 Cluster status messages

Status message	Description
`mode: enabled: cluster active - MASTER` `master: mozart`	The node is the master.
`mode: enabled: cluster active - SLAVE` `master: mozart`	The node is a slave.
`mode: enabled: cluster active - role not set` `master: mozart` `state: joining` `reconfig: master update`	The node has not yet been assigned a role, and is in the process of joining the cluster.
`mode: enabled: cluster active - SLAVE` `master: mozart` `state: joining`	The node is configured as a slave, and is in the process of joining the cluster.
`mode: enabled: cluster inactive`	The cluster is not active on this node.

Note: If the `vxconfigd` daemon is disabled, no cluster information is displayed.

See the `vxdctl`(1M) manual page for more information.

Determining if a disk is shareable

The `vxdisk` utility manages VxVM disks. To use the `vxdisk` utility to determine whether a disk is part of a cluster-shareable disk group, use the following command:

```
# vxdisk list accessname
```

where *accessname* is the disk access name (or device name). A portion of the output from this command (for the device `c4t1d0`) is shown here:

```
Device:     c4t1d0
devicetag:  c4t1d0
type:       auto
clusterid:  cvm2
disk:       name=shdg01 id=963616090.1034.cvm2
timeout:    30
group:      name=shdg id=963616065.1032.cvm2
flags:      online ready autoconfig shared imported
...
```

Note that the `clusterid` field is set to `cvm2` (the name of the cluster), and the `flags` field includes an entry for `shared`. When a node is not joined to the cluster, the `flags` field contains the `autoimport` flag instead of `imported`.

Listing shared disk groups

`vxdg` can be used to list information about shared disk groups. To display information for all disk groups, use the following command:

```
# vxdg list
```

Example output from this command is displayed here:

```
NAME         STATE            ID
rootdg       enabled          774215886.1025.teal
group2       enabled,shared   774575420.1170.teal
group1       enabled,shared   774222028.1090.teal
```

Shared disk groups are designated with the flag `shared`.

To display information for shared disk groups only, use the following command:

```
# vxdg -s list
```

Example output from this command is as follows:

```
NAME         STATE            ID
group2       enabled,shared   774575420.1170.teal
group1       enabled,shared   774222028.1090.teal
```

To display information about one specific disk group, use the following command:

```
# vxdg list diskgroup
```

The following is example output for the command `vxdg list group1` on the master:

```
Group:     group1
dgid:      774222028.1090.teal
import-id: 32768.1749
flags:     shared
version:   140
alignment: 8192 (bytes)
ssb:                    on
local-activation: exclusive-write
cluster-actv-modes: node0=ew node1=off
detach-policy: local
private_region_failure: leave
copies:    nconfig=2 nlog=2
config:    seqno=0.1976 permlen=1456 free=1448 templen=6
loglen=220
config disk c1t0d0 copy 1 len=1456 state=clean online
config disk c1t0d0 copy 1 len=1456 state=clean online
log disk c1t0d0 copy 1 len=220
log disk c1t0d0 copy 1 len=220
```

Note that the `flags` field is set to `shared`. The output for the same command when run on a slave is slightly different. The `local-activation` and `cluster-actv-modes` fields display the activation mode for this node and for each node in the cluster respectively. The `detach-policy` and `private_region_failure` fields indicate how the cluster behaves in the event of loss of connectivity to the disks, and to the configuration and log copies on the disks.

Creating a shared disk group

Note: Shared disk groups can only be created on the master node.

If the cluster software has been run to set up the cluster, a shared disk group can be created using the following command:

```
# vxdg -s init diskgroup [diskname=]devicename
```

where *diskgroup* is the disk group name, *diskname* is the administrative name chosen for a VM disk, and *devicename* is the device name (or disk access name).

Caution: The operating system cannot tell if a disk is shared. To protect data integrity when dealing with disks that can be accessed by multiple systems, use the correct designation when adding a disk to a disk group. VxVM allows you to add a disk that is not physically shared to a shared disk group if the node where the disk is accessible is the only node in the cluster. However, this means that other nodes cannot join the cluster. Furthermore, if you attempt to add the same disk to different disk groups (private or shared) on two nodes at the same time, the results are undefined. Perform all configuration on one node only, and preferably on the master node.

Importing disk groups as shared

Note: Shared disk groups can only be imported on the master node.

Disk groups can be imported as shared using the `vxdg -s import` command. If the disk groups are set up before the cluster software is run, the disk groups can be imported into the cluster arrangement using the following command:

```
# vxdg -s import diskgroup
```

where *diskgroup* is the disk group name or ID. On subsequent cluster restarts, the disk group is automatically imported as shared. Note that it can be necessary to deport the disk group (using the `vxdg deport` *diskgroup* command) before invoking the `vxdg` utility.

Forcibly importing a disk group

You can use the `-f` option to the `vxdg` command to import a disk group forcibly.

Caution: The force option(`-f`) must be used with caution and only if you are fully aware of the consequences such as possible data corruption.

When a cluster is restarted, VxVM can refuse to auto-import a disk group for one of the following reasons:

- A disk in the disk group is no longer accessible because of hardware errors on the disk. In this case, use the following command to forcibly reimport the disk group:

  ```
  # vxdg -s -f import diskgroup
  ```

 Note: After a forced import, the data on the volumes may not be available and some of the volumes may be in the disabled state.

- Some of the nodes to which disks in the disk group are attached are not currently in the cluster, so the disk group cannot access all of its disks. In this case, a forced import is unsafe and must not be attempted because it can result in inconsistent mirrors.

Converting a disk group from shared to private

Note: Shared disk groups can only be deported on the master node.

To convert a shared disk group to a private disk group, first deport it on the master node using this command:

```
# vxdg deport diskgroup
```

Then reimport the disk group on any cluster node using this command:

```
# vxdg import diskgroup
```

Moving objects between disk groups

As described in "Moving objects between disk groups" on page 203, you can use the `vxdg move` command to move a self-contained set of VxVM objects such as disks and top-level volumes between disk groups. In a cluster, you can move such objects between private disk groups on any cluster node where those disk groups are imported.

Note: You can only move objects between shared disk groups on the master node. You cannot move objects between private and shared disk groups.

Splitting disk groups

As described in "Splitting disk groups" on page 205, you can use the `vxdg split` command to remove a self-contained set of VxVM objects from an imported disk group, and move them to a newly created disk group.

Splitting a private disk group creates a private disk group, and splitting a shared disk group creates a shared disk group. You can split a private disk group on any cluster node where that disk group is imported. You can only split a shared disk group or create a shared target disk group on the master node.

See "Moving objects between disk groups" on page 203.

Joining disk groups

As described in "Joining disk groups" on page 206, you can use the `vxdg join` command to merge the contents of two imported disk groups. In a cluster, you

can join two private disk groups on any cluster node where those disk groups are imported.

If the source disk group and the target disk group are both shared, you must perform the join on the master node.

Note: You cannot join a private disk group and a shared disk group.

Changing the activation mode on a shared disk group

Note: The activation mode for access by a cluster node to a shared disk group is set on that node.

The activation mode of a shared disk group can be changed using the following command:

```
# vxdg -g diskgroup set activation=mode
```

The activation *mode* is one of `exclusivewrite` or `ew`, `readonly` or `ro`, `sharedread` or `sr`, `sharedwrite` or `sw`, or `off`.

If you use this command to change the activation mode of a shared disk group, you must first change the activation mode to `off` before setting it to any other value, as shown here:

```
# vxdg -g myshdg set activation=off
# vxdg -g myshdg set activation=readonly
```

See "Activation modes of shared disk groups" on page 402.

Setting the disk detach policy on a shared disk group

Note: The disk detach policy for a shared disk group can only be set on the master node.

The `vxdg` command may be used to set either the `global` or `local` disk detach policy for a shared disk group:

```
# vxdg -g diskgroup set diskdetpolicy=global|local
```

The default disk detach policy is `global`.

See "Connectivity policy of shared disk groups" on page 404.

Setting the disk group failure policy on a shared disk group

Note: The disk group failure policy for a shared disk group can only be set on the master node.

The `vxdg` command may be used to set either the `dgdisable` or `leave` failure policy for a shared disk group:

```
# vxdg -g diskgroup set dgfailpolicy=dgdisable|leave
```

The default failure policy is `dgdisable`.

See "Disk group failure policy" on page 407.

Creating volumes with exclusive open access by a node

Note: All shared volumes, including those with exclusive open access, can only be created on the master node.

When using the `vxassist` command to create a volume, you can use the `exclusive=on` attribute to specify that the volume may only be opened by one node in the cluster at a time. For example, to create the mirrored volume `volmir` in the disk group `dskgrp`, and configure it for exclusive open, use the following command:

```
# vxassist -g dskgrp make volmir 5g layout=mirror exclusive=on
```

Multiple opens by the same node are also supported. Any attempts by other nodes to open the volume fail until the final close of the volume by the node that opened it.

Specifying `exclusive=off` instead means that more than one node in a cluster can open a volume simultaneously. This is the default behavior.

Setting exclusive open access to a volume by a node

Note: Exclusive open access on a volume can only be set on the master node. Ensure that none of the nodes in the cluster have the volume open when setting this attribute.

You can set the `exclusive=on` attribute with the vxvol command to specify that an existing volume may only be opened by one node in the cluster at a time.

For example, to set exclusive open on the volume `volmir` in the disk group `dskgrp`, use the following command:

```
# vxvol -g dskgrp set exclusive=on volmir
```

Multiple opens by the same node are also supported. Any attempts by other nodes to open the volume fail until the final close of the volume by the node that opened it.

Specifying `exclusive=off` instead means that more than one node in a cluster can open a volume simultaneously. This is the default behavior.

Displaying the cluster protocol version

The following command displays the cluster protocol version running on a node:

```
# vxdctl list
```

This command produces output similar to the following:

```
Volboot file
version: 3/1
seqno: 0.19
cluster protocol version: 70
hostid: giga
entries:
```

You can also check the existing cluster protocol version using the following command:

```
# vxdctl protocolversion
```

This produces output similar to the following:

```
Cluster running at protocol 70
```

Displaying the supported cluster protocol version range

The following command displays the maximum and minimum protocol version supported by the node and the current protocol version:

```
# vxdctl support
```

This command produces out put similar to the following:

```
Support information:
  vxconfigd_vrsn:      21
  dg_minimum:          20
  dg_maximum:          140
  kernel:              15
  protocol_minimum:    40
  protocol_maximum:    70
  protocol_current:    70
```

You can also use the following command to display the maximum and minimum cluster protocol version supported by the current Veritas Volume Manager release:

```
# vxdctl protocolrange
```

This produces output similar to the following:

```
minprotoversion: 40, maxprotoversion: 70
```

Upgrading the cluster protocol version

Note: The cluster protocol version can only be updated on the master node.

After all the nodes in the cluster have been updated with a new cluster protocol, you can upgrade the entire cluster using the following command on the master node:

```
# vxdctl upgrade
```

Recovering volumes in shared disk groups

Note: Volumes can only be recovered on the master node.

The `vxrecover` utility is used to recover plexes and volumes after disk replacement. When a node leaves a cluster, it can leave some mirrors in an inconsistent state. The `vxrecover` utility can be used to recover such volumes. The `-c` option to `vxrecover` causes it to recover all volumes in shared disk groups. The `vxconfigd` daemon automatically calls the `vxrecover` utility with the `-c` option when necessary.

Note: While the `vxrecover` utility is active, there can be some degradation in system performance.

Obtaining cluster performance statistics

The `vxstat` utility returns statistics for specified objects. In a cluster environment, `vxstat` gathers statistics from all of the nodes in the cluster. The statistics give the total usage, by all nodes, for the requested objects. If a local object is specified, its local usage is returned.

You can optionally specify a subset of nodes using the following form of the command:

```
# vxstat -g diskgroup -n node[,node...]
```

where *node* is the CVM node ID number. You can find out the CVM node ID by using the following command:

```
# vxclustadm nidmap
```

If a comma-separated list of nodes is supplied, the `vxstat` utility displays the sum of the statistics for the nodes in the list.

For example, to obtain statistics for node 2, volume `vol1`, use the following command:

```
# vxstat -g group1 -n 2 vol1
```

This command produces output similar to the following:

```
                OPERATIONS      BLOCKS            AVG TIME(ms)
TYP  NAME   READ    WRITE   READ      WRITE   READ    WRITE
vol  vol1   2421    0       600000    0       99.0    0.0
```

To obtain and display statistics for the entire cluster, use the following command:

```
# vxstat -b
```

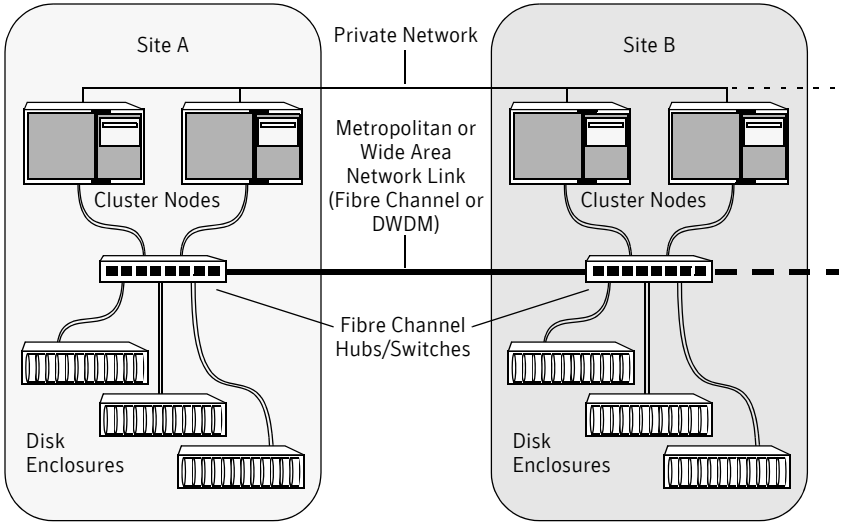
The statistics for all nodes are summed. For example, if node 1 performed 100 I/O operations and node 2 performed 200 I/O operations, `vxstat -b` displays a total of 300 I/O operations.

Chapter 14

Administering sites and remote mirrors

In a Remote Mirror configuration (also known as a campus cluster or stretch cluster) the hosts and storage of a cluster that would usually be located in one place, are instead divided between two or more sites. These sites are typically connected via a redundant high-capacity network that provides access to storage and private link communication between the cluster nodes. A typical two-site remote mirror configuration is illustrated in Figure 14-1.

Figure 14-1 Example of a two-site remote mirror configuration



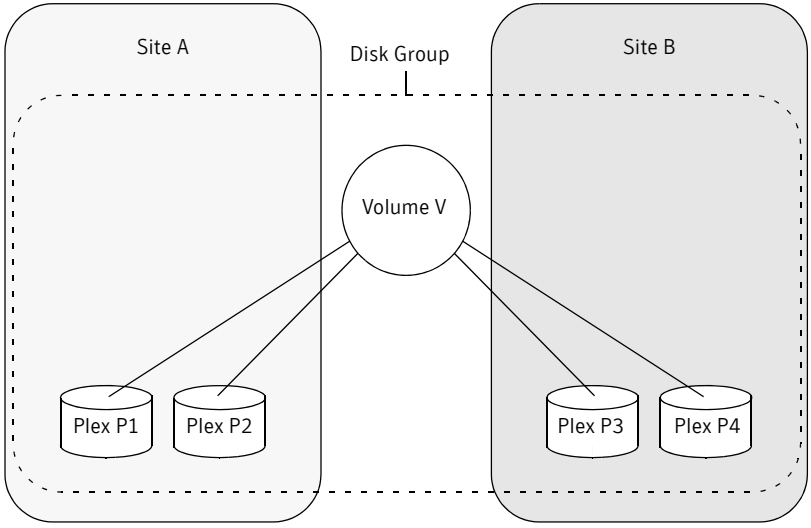
If a disk group is configured across the storage at the sites, and inter-site communication is disrupted, there is a possibility of a serial split brain condition arising if each site continues to update the local disk group configuration copies (see “Handling conflicting configuration copies” on page 190). VxVM provides mechanisms for dealing with the serial split brain condition, monitoring the health of a remote mirror, and testing the robustness of the cluster against various types of failure (also known as *fire drill*).

For applications and services to function correctly at a site when other sites have become inaccessible, at least one complete plex of each volume must be configured at each site (*site-based allocation*), and the consistency of the data in the plexes at each site must be ensured (*site consistency*).

By tagging disks with site names, storage can be allocated from the correct location when creating, resizing or relocating a volume, and when changing a volume’s layout.

Figure 14-2 shows an example of a site-consistent volume with two plexes configured at each of two sites. The storage for plexes P1 and P2 is allocated storage that is tagged as belonging to site A, and the storage for plexes P3 and P4 is allocated storage that is tagged as belonging to site B.

Figure 14-2 Site-consistent volume with two plexes at each of two sites



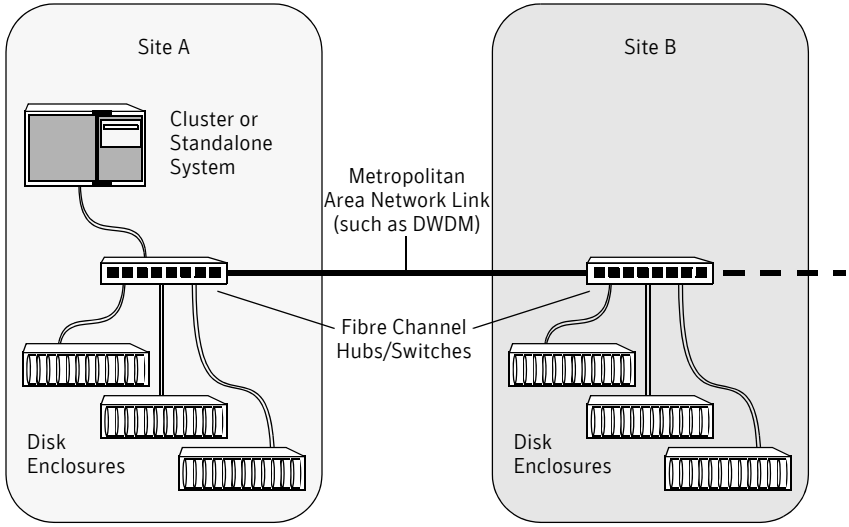
Although not shown in this figure, DCO log volumes are also mirrored across the sites, and disk group configuration copies are distributed across the sites.

To enhance read performance, VxVM will service reads from the plexes at the local site where an application is running if the `siteread` read policy is set on a volume. Writes are written to plexes at all sites.

The site consistency of a volume is ensured by detaching a site when its last complete plex fails at that site. If a site fails, all its plexes are detached and the site is said to be detached.

Configurations with remote storage only are also supported as illustrated in Figure 14-3.

Figure 14-3 Example of a two-site configuration with remote storage only



Configuring sites for hosts and disks

Note: The Remote Mirror feature requires that the Site Awareness license has been installed on all hosts at all sites that are participating in the configuration.

Use the following command to set the site name for each host:

```
# vxdctl set site=sitename
```

The name that has been assigned to a site is stored in the `/etc/vx/volboot` file, and can be displayed by using the `vxdctl list` command:

```
# vxdctl list | grep siteid
siteid: building1
```

To remove the site name from a host, use this command:

```
# vxdctl [-F] unset site
```

The `-F` option is required if any imported disk groups are registered to the site.

To tag disks with a site name, use the `vxdisk settag` command as shown here:

```
# vxdisk [-g diskgroup] settag disk site=sitename
```

where the disk can be specified either by the disk access name or the disk media name. You must repeat this command for each of the disks that are to be registered to a site.

To check which disks are registered to a site, use the `vxdisk listtag` command:

```
# vxdisk [-g diskgroup] listtag
```

To remove the site tag from a disk, use the `vxdisk rmtag` command:

```
# vxdisk rmtag disk site=sitename
```

Configuring site-based allocation on a disk group

To turn on the site-based allocation requirement for a site that is registered to a disk group, use the `vxdg addsite` command for each site at which site-based allocation is required:

```
# vxdg -g diskgroup [-f] addsite sitename
```

Each volume on which the `allsites` attribute is set to `on` is checked to ensure that it has at least one plex at each site, and the command fails if this condition is not met. If the `-f` option is specified, the command does not fail, but instead it sets the `allsites` attribute for the volume to `off`.

Provided that the Site Awareness license is installed on all the hosts in the Remote Mirror configuration, a volume is automatically mirrored across sites and its read policy is set to `siteread` when it is created. If site-based allocation is not required, or is not possible (as is the case for RAID-5 volumes), specify the `allsites=off` attribute to the `vxassist` command.

To remove the site-based allocation requirement from a site, use this command:

```
# vxdg -g diskgroup [-f] rmsite sitename
```

The -f option allows the requirement to be removed if the site is detached or offline.

The site name is not removed from the disks. If required, use the vxdisk rmtag command to remove the site tag as described in "Configuring sites for hosts and disks" on page 434.

Configuring site consistency on a disk group

Having configured site-based allocation on a disk group as described in "Configuring site-based allocation on a disk group" on page 434, turn on the site consistency requirement for a disk group by using the vxdg command:

```
# vxdg -g diskgroup set siteconsistent=on
```

Note: This command does not set the siteconsistent attribute on existing volumes in the disk group. Although newly created volumes inherit the setting from the disk group, it does not change when the disk group setting is changed. See "Configuring site consistency on a volume" on page 435.

All the disks in a disk group must be registered to one of the sites before you can set the siteconsistent attribute on the disk group.

To verify whether site consistency has been enabled for a disk group, use the following command:

```
# vxdg list diskgroup | grep siteconsistent
flags: siteconsistent
```

To turn off the site consistency requirement for a disk group, use the following command:

```
# vxdg -g diskgroup set siteconsistent=off
```

Configuring site consistency on a volume

To set the site consistency requirement when creating a volume, specify the siteconsistent attribute to the vxassist make command, for example:

```
# vxassist [-g diskgroup] make volume size \
  nmirror=4 siteconsistent={on|off}
```

By default, a volume inherits the value that is set on its disk group.

Note: By default, creating a site-consistent volume also creates an associated version 20 DCO volume, and enables Persistent FastResync on the volume. This allows faster recovery of the volume during the reattachment of a site.

To turn on the site consistency requirement for an existing volume, use the following form of the vxvol command:

```
# vxvol [-g diskgroup] set siteconsistent=on volume
```

To turn off the site consistency requirement for a volume, use the following command:

```
# vxvol [-g diskgroup] set siteconsistent=off volume
```

Note: The siteconsistent and allsites attributes must be set to off for RAID-5 volumes in a site-consistent disk group.

Setting the siteread policy on a volume

If the Site Awareness license is installed on all the hosts in the Remote Mirror configuration, the disk group is configured for site consistency with several sites enabled, and the allsites=on attribute is specified for a volume, the default read policy is siteread.

If required, you can use the following command to set the siteread policy on a volume:

```
# vxvol [-g diskgroup] rdpol siteread volume
```

This command has no effect if a site name has not been set for the host.

See "Changing the read policy for mirrored volumes" on page 289.

Site-based allocation of storage to volumes

The vxassist command can be used to create volumes only from storage that exists at a specified site, as shown in this example:

```
# vxassist -g diskgroup make volume size site:site1 \
  [allsites={on|off}] [siteconsistent={on|off}]
```

The storage class site is used in similar way to other storage classes with the vxassist command, such as enclr, ctlr and disk.

See "Mirroring across targets, controllers or enclosures" on page 255.

Note: If the Site Awareness license is installed on all the hosts in the Remote Mirror configuration, and site consistency is enabled on a volume, the `vxassist` command attempts to allocate storage across the sites that are registered to a disk group. If not enough storage is available at all sites, the command fails unless you also specify the `allsites=off` attribute.

By default, the `allsites` attribute is set to `on` for volume in a site-consistent disk group. The `allsites` and `siteconsistent` attributes must be set to `off` for RAID-5 volumes in a site-consistent disk group.

In a similar way to mirroring across controllers, you can also ensure that plexes are created at all sites that are registered for a disk group:

```
# vxassist -g diskgroup make volume size mirror=site
```

The `allsites` and `siteconsistent` attributes can be combined to create a non-site-consistent mirrored volume with plexes only at some of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  site:site1 site:site2 ... allsites=off siteconsistent=off
```

a non-site-consistent mirrored volume with plexes at all of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  allsites=on siteconsistent=off
```

a site-consistent mirrored volume with plexes only at some of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  site:site1 site:site2 ... allsites=off siteconsistent=on
```

or a site-consistent mirrored volume with plexes at all of the sites:

```
# vxassist -g diskgroup make volume size mirror=site \
  allsites=on siteconsistent=on
```

By default, the number of mirrors created is equal to the number of sites that are configured in the disk group. There should be a minimum of 2 plexes at each site. To override the default behavior, use the `nmirror` attribute to specify the total number of mirrors:

```
# vxassist -g diskgroup make volume size mirror=site \
  nmirror=4 site:site1 site:site2 ... [allsites={on|off}] \
  [siteconsistent={on|off}]
```

If a volume is intended to be site consistent, the number of plexes that are specified must be a multiple of the number of sites.

Examples of storage allocation using sites

The examples in the following table demonstrate how to use site names with the `vxassist` command to allocate storage. The disk group, `ccdg`, has been enabled for site consistency with disks configured at two sites, `site1` and `site2`.

Command	Description
`# vxassist -g ccdg make vol 2g \ nmirror=2`	Create a volume with one mirror at each site. The value of `nmirror` must be greater than or equal to the number of sites unless `allsites` is set to `off`.
`# vxassist -g ccdg -o ordered \ make vol 2g \ layout=mirror-stripe ncol=3 \ ccdg01 ccdg02 ccdg03 ccdg09 \ ccdg10 ccdg11`	Create a mirrored-stripe volume specifying allocation order to validate redundancy across the sites. The named disks must be tagged with the appropriate site name, and there must be sufficient disks at each site to create the volume.
`# vxassist -g ccdg make vol 2g \ nmirror=2 mirror=site \ site:site1 site:site2`	Create a volume with one mirror at each of the named sites. All sites must be named and the value of `nmirror` must be greater than or equal to the number of sites unless `allsites` is set to `off`.
`# vxassist -g ccdg make vol 2g \ nmirror=2 ccdg01 ccdg09`	Create a volume with one mirror on each of the named disks. The named disks must be tagged with the appropriate site name, and there must be sufficient disks at each site to create the volume.
`# vxassist -g ccdg make vol 2g \ nmirror=2 siteconsistent=off \ allsites=off`	Create a mirrored volume that is not site consistent. Both mirrors can be allocated from any available storage in the disk group.
`# vxassist -g ccdg make vol 2g \ nmirror=2 site:site2 \ siteconsistent=off \ allsites=off`	Create a mirrored volume that is not site consistent. Both mirrors are allocated from any available storage in the disk group that is tagged as belonging to `site2`.
`# vxassist -g ccdg make vol 2g \ nmirror=2 !site:site1 \ siteconsistent=off \ allsites=off`	Create a mirrored volume that is not site consistent. Both mirrors are allocated from any available storage in the disk group that is tagged as not belonging to `site1`.
`# vxassist -g ccdg mirror vol \ site:site1`	Add a mirror at a specified site. The command fails if there is insufficient storage available at the site.

Command	Description
`# vxassist -g ccdg remove \ mirror vol site:site1`	Remove a mirror from a volume at a specified site. If the volume is site consistent, the command fails if this would remove the last remaining plex at a site.
`# vxassist -g ccdg growto vol 4g`	Grow a volume. If the volume is site consistent, the command fails if there is insufficient storage available at any site.

Making an existing disk group site consistent

To make an existing disk group site consistent

1. Ensure that the disk group is updated to at least version 140, by running the `vxdg upgrade` command on it:

   ```
   # vxdg upgrade diskgroup
   ```

2. On each host that can access the disk group, define the site name:

   ```
   # vxdctl set site=sitename
   ```

3. Tag all the disks in the disk group with the appropriate site name:

   ```
   # vxdisk [-g diskgroup] settag disk site=sitename
   ```

4. Use the `vxdg move` command to move any unsupported RAID-5 volumes to another disk group. Alternatively, use the `vxassist convert` command to convert the volumes to a supported layout such as `mirror` or `mirror-stripe`. You can use the `site` and `mirror=site` storage allocation attribute to ensure that the plexes are created on the correct storage.

5. Use the `vxevac` command to ensue that the volumes have equal number of plexes at each site. You can use the `site` and `mirror=site` storage allocation attribute to ensure that the plexes are created on the correct storage.

6. Register a site record for each site with the disk group:

   ```
   # vxdg -g diskgroup addsite sitename
   ```

7. Turn on site consistency for the disk group:

   ```
   # vxdg -g diskgroup set siteconsistent=on
   ```

8. Turn on site consistency for each volume in the disk group:

   ```
   # vxvol [-g diskgroup] set siteconsistent=on volume ...
   ```

Fire drill — testing the configuration

Caution: To avoid potential loss of service or data, it is recommended that you do not use these procedures on a live system.

After validating that the consistency of the volumes and disk groups at your sites, you should validate the procedures that you will use in the event of the various possible types of failure. A fire drill allows you to test that a site can be brought up cleanly during recovery from a disaster scenario such as site failure.

Simulating site failure

To simulate the failure of a site, use the following command to detach all the devices at a specified site:

```
# vxdg -g diskgroup [-f] detachsite sitename
```

The `-f` option must be specified if any plexes configured on storage at the site are currently online.

Recovery from simulated site failure

Use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup [-o overridessb] reattachsite sitename
# vxrecover -g diskgroup
```

It may be necessary to specify the `-o overridessb` option if a serial split-brain condition is indicated.

Automatic site reattachment

The site reattachment daemon, `vxsited`, provides automatic reattachment of sites. `vxsited` uses the `vxnotify` mechanism to monitor storage coming back online on a site after a previous failure, and to restore redundancy of mirrors across sites.

If the hot-relocation daemon, `vxrelocd`, is running, `vxsited` attempts to reattach the site, and allows `vxrelocd` to try to use the available disks in the disk group to relocate the failed subdisks. If `vxrelocd` succeeds in relocating the failed subdisks, it starts the recovery of the plexes at the site. When all the plexes have been recovered, the plexes are put into the ACTIVE state, and the state of the site is set to ACTIVE.

If `vxrelocd` is not running, `vxsited` reattaches a site only when all the disks at that site become accessible. After reattachment succeeds, `vxsited` sets the

site state to ACTIVE, and initiates recovery of the plexes. When all the plexes have been recovered, the plexes are put into the ACTIVE state.

Note: `vxsited` does not try to reattach a site that you have explicitly detached by using the `vxdg detachsite` command.

The automatic site reattachment feature is enabled by default. The `vxsited` daemon uses email to notify `root` of any attempts to reattach sites and to initiate recovery of plexes at those sites. To send mail to other users, add the user name to the line that starts `vxsited` in the `/etc/init.d/vxvm-recover` startup script, and reboot the system.

If you do not want a site to be recovered automatically, kill the `vxsited` daemon, and prevent it from restarting. To kill the daemon, run the following command from the command line:

```
# ps -afe
```

Locate the process table entry for `vxsited`, and kill it by specifying its process ID:

```
# kill -9 PID
```

If there is no entry in the process table for `vxsited`, the automatic site reattachment feature is disabled.

To prevent the automatic site reattachment feature from being restarted, comment out the line that starts `vxsited` in the `/etc/init.d/vxvm-recover` startup script.

Failure scenarios and recovery procedures

The site reattachment daemon will provide automatic reattachment of detached sites. See "Automatic site reattachment" on page 440. This section describes the procedures that you can follow if automatic reattachment does not take place. For example, this can happen if you have disable automatic reattachment, or if you manually detach a site.

Possible failure scenarios and recovery techniques are listed in the following table:

Failure scenario	Recovery technique
Disruption of network link between sites.	See "Recovery from a loss of site connectivity" on page 442.
Failure of hosts at a site.	See "Recovery from host failure" on page 442.

Failure scenario	Recovery technique
Failure of storage at a site.	See “Recovery from storage failure” on page 442.
Failure of both hosts and storage at a site.	See “Recovery from site failure” on page 443.

Recovery from a loss of site connectivity

If the network links between the sites are disrupted, the application environments may continue to run in parallel, and this may lead to inconsistencies between the disk group configuration copies at the sites. When connectivity between the sites is restored, a serial split-brain condition may then exist between the sites. One site must be chosen as having the preferred version of the disk group configuration copies. The configuration copies at the other sites can then be updated from these copies.

You can use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup -o overridessb reattachsite sitename
# vxrecover -g diskgroup
```

In the case that the host systems are configured at a single site with only storage at the remote sites, the usual resynchronization mechanism of VxVM is used to recover the remote plexes when the storage comes back on line.

Recovery from host failure

If one or more cluster nodes fail at a site, but the storage remains online, this is handled either by VCS failover in the case of the Storage Foundation HA product, or by node takeover in the case that the node was the master for a shared disk group as supported by the Storage Foundation Cluster File System software.

Recovery from storage failure

If storage fails at a site, the plexes that are configured on that storage are detached locally if a site-consistent volume still has other mirrors available at the site. The hot-relocation feature of VxVM will attempt to recreate the failed plexes on other available storage in the disk group. If no plexes of a site-consistent volume remain in operation at a site, and hot-relocation cannot recreate the plexes at that site, the site is detached. Because site connectivity has not been lost, applications running on hosts at the site can still access data

at the other sites. When the storage comes back online, you can use the following commands to reattach a site and recover the disk group:

```
# vxdg -g diskgroup reattachsite sitename
# vxrecover -g diskgroup
```

Recovery from site failure

If all the hosts and storage fail at a site, you can use the following commands to reattach the site after it comes back online, and to recover the disk group:

```
# vxdg -g diskgroup [-o overridessb] reattachsite sitename
# vxrecover -g diskgroup
```

The `-o overridessb` option is only required if a serial split-brain condition is indicated, which may happen if the site was brought back up while the private network link was inoperative. This option updates the configuration database on the reattached site with the consistent copies at the other sites.

Chapter 15

Using Storage Expert

About Storage Expert

System administrators often find that gathering and interpreting data about large and complex configurations can be a difficult task. Veritas Storage Expert is designed to help in diagnosing configuration problems with VxVM.

Storage Expert consists of a set of simple commands that collect VxVM configuration data and compare it with "best practice." Storage Expert then produces a summary report that shows which objects do not meet these criteria and makes recommendations for VxVM configuration improvements.

These user-configurable tools help you as an administrator to verify and validate systems and non-optimal configurations in both small and large VxVM installations.

See "How Storage Expert works" on page 446.

See "Before using Storage Expert" on page 446.

See "Running Storage Expert" on page 446.

See "Identifying configuration problems using Storage Expert" on page 449.

See "Rule definitions and attributes" on page 456.

See the `vxse`(1M) manual page.

How Storage Expert works

Storage Expert components include a set of rule scripts and a rules engine. The rules engine runs the scripts and produces ASCII output, which is organized and archived by Storage Expert's report generator. This output contains information about areas of VxVM configuration that do not meet the set criteria. By default, output is sent to the screen, but you can send it to a file using standard output redirection.

Before using Storage Expert

Storage Expert is included in the `VRTSvxvm` package. Even if you do not plan to use the VEA graphical user interface, you must also have installed the following packages to run vxse:

- `VRTSob`
- `VRTSvmpro`

The VEA service must also be started on the system by running the command `/opt/VRTS/bin/vxsvc`.

Running Storage Expert

The executable rule files are located in the directory, `/opt/VRTS/vxse/vxvm`. It is recommended that this directory be added to the `PATH` variable.

The rules are invoked using the following command-line syntax:

```
# rulename [options] keyword [attribute=value ...]
```

Each of the rules performs a different function.

See "Rule definitions and attributes" on page 456.

The following options may be specified:

`-d` *defaults_file*	Specifies an alternate defaults file.
`-g` *diskgroup*	Specifies verbose output format.
`-v`	Specifies the disk group to be examined.

One of the following keywords must be specified:

`check`	Lists the default values used by the rule's attributes.

`info`	Describes what the rule does.
`list`	Lists the attributes of the rule that you can set.
`run`	Runs the rule.

See "Rule definitions and attributes" on page 456.

Discovering what a rule does

To obtain details about what a rule does, use the `info` keyword, as in the following example:

```
# vxse_stripes2 info
VxVM vxse:vxse_stripes2 INFO V-5-1-6153 This rule checks for
stripe volumes which have too many or too few columns
```

Displaying rule attributes and their default values

To see the attributes that are available for a given rule, use the `list` keyword. In the following example, the single attribute, `mirror_threshold`, is shown for the rule `vxse_drl1`:

```
# vxse_drl1 list
VxVM vxse:vxse_drl1 INFO V-5-1-6004
vxse_drl1 - TUNEABLES default values
-------------------------------------------------------
      mirror_threshold - large mirror threshold size
                         Warn if a mirror is of greater
                         than this size and does not have
                         an attached DRL log.
```

To see the default values of a specified rule's attributes, use the `check` keyword as shown here:

```
# vxse_stripes2 check
vxse_stripes2 - TUNEABLES
-------------------------------------------------------
VxVM vxse:vxse_stripes2 INFO V-5-1-5546
        too_wide_stripe - (16) columns in a striped volume
        too_narrow_stripe - (3) columns in a striped volume
```

Storage Expert lists the default value of each of the rule's attributes.

See "Rule definitions and attributes" on page 456.

To alter the behavior of rules, you can change the value of their attributes.

See "Setting rule attributes" on page 448.

Running a rule

The `run` keyword invokes a default or reconfigured rule on a disk group or file name, for example:

```
# vxse_dg1 -g mydg run
VxVM vxse:vxse_dg1 INFO V-5-1-5511 vxse_vxdg1 - RESULTS
------------------------------------------------------------

vxse_dg1 PASS:
Disk group (mydg) okay amount of disks in this disk group (4)
```

This indicates that the specified disk group (`mydg`) met the conditions specified in the rule.

See "Rule result types" on page 448.

You can set Storage Expert to run as a `cron` job to notify administrators, and to archive reports automatically.

Rule result types

Running a rule generates output that shows the status of the objects that have been examined against the rule:

`INFO`	Information about the specified object; for example "RAID-5 does not have a log."
`PASS`	The object met the conditions of the rule.
`VIOLATION`	The object did not meet the conditions of the rule.

Setting rule attributes

You can set attributes in the following ways:

- Enter an attribute on the command line, for example:

  ```
  # vxse_drl2 -g mydg run large_mirror_size=30m
  ```

- Create your own defaults file, and specify that file on the command line:

  ```
  # vxse_drl2 -d mydefaultsfile run
  ```

 Lines in this file contain attribute values definitions for a rule in this format:

 rule_name,attribute=value

 For example, the following entry defines a value of 20 gigabytes for the attribute `large_mirror_size` of the rule `vxse_drl2`:

  ```
  vxse_drl2,large_mirror_size=20g
  ```

 You can specify values that are to be ignored by inserting a # character at the start of the line, for example:

  ```
  #vxse_drl2,large_mirror_size=20g
  ```

- Edit the attribute values that are defined in the `/etc/default/vxse` file. If you do this, make a backup copy of the file in case you need to regress your changes.

Attributes are applied using the following order of precedence from highest to lowest:

- A value specified on the command line.
- A value specified in a user-defined defaults file.
- A value in the `/etc/default/vxse` file that has not been commented out.
- A built-in value defined at compile time.

Identifying configuration problems using Storage Expert

Storage Expert provides a large number of rules that help you to diagnose configuration issues that might cause problems for your storage environment. Each rule describes the issues involved, and suggests remedial actions.

The rules help you to diagnose problems in the following categories:

- Recovery time
- Disk groups
- Disk striping
- Disk sparing and relocation management
- Hardware failures
- Rootability
- System name

See "Rule definitions and attributes" on page 456.

Recovery time

Several "best practice" rules enable you to check that your storage configuration has the resilience to withstand a disk failure or a system failure.

Checking for multiple RAID-5 logs on a physical disk (vxse_disklog)

To check whether more than one RAID-5 log exists on the same physical disk, run rule `vxse_disklog`.

RAID-5 log mirrors for the same physical volume should be located on separate physical disks to ensure redundancy. More than one RAID-5 log on a disk also makes the recovery process longer and more complicated.

Checking for large mirror volumes without a dirty region log (vxse_drl1)

To check whether large mirror volumes (larger than 1GB) have an associated dirty region log (DRL), run rule `vxse_drl1`.

Creating a DRL speeds recovery of mirrored volumes after a system crash. A DRL tracks those regions that have changed and uses the tracking information to recover only those portions of the volume that need to be recovered. Without a DRL, recovery is accomplished by copying the full contents of the volume between its mirrors. This process is lengthy and I/O intensive.

See "Preparing a volume for DRL and instant snapshots" on page 275.

Checking for large mirrored volumes without a mirrored dirty region log (vxse_drl2)

To check whether a large mirrored volume has a mirrored DRL log, run rule `vxse_drl2`.

Mirroring the DRL log provides added protection in the event of a disk failure.

See "Preparing a volume for DRL and instant snapshots" on page 275.

Checking for RAID-5 volumes without a RAID-5 log (vxse_raid5log1)

To check whether a RAID-5 volume has an associated RAID-5 log, run rule `vxse_raid5log1`.

In the event of both a system failure and a failure of a disk in a RAID-5 volume, data that is not involved in an active write could be lost or corrupted if there is no RAID-5 log.

See "Adding a RAID-5 log" on page 283.

Checking minimum and maximum RAID-5 log sizes (vxse_raid5log2)

To check that the size of RAID-5 logs falls within the minimum and maximum recommended sizes, run rule `vxse_raid5log2`.

The recommended minimum and maximum sizes are 64MB and 1GB respectively. If `vxse_raid5log2` reports that the size of the log is outside these boundaries, adjust the size by replacing the log.

Checking for non-mirrored RAID-5 logs (vxse_raid5log3)

To check that the RAID-5 log of a large volume is mirrored, run the `vxse_raid5log3` rule.

A mirror of the RAID-5 log protects against loss of data due to the failure of a single disk. You are strongly advised to mirror the log if `vxse_raid5log3` reports that the log of a large RAID-5 volume does not have a mirror.

See "Adding a RAID-5 log" on page 283.

Disk groups

Disks groups are the basis of VxVM storage configuration so it is critical that the integrity and resilience of your disk groups are maintained. Storage Expert provides a number of rules that enable you to check the status of disk groups and associated objects.

Checking whether a configuration database is too full (vxse_dg1)

To check whether the disk group configuration database has become too full, run rule `vxse_dg1`.

By default, this rule suggests a limit of 250 for the number of disks in a disk group. If one of your disk groups exceeds this figure, you should consider creating a new disk group. The number of objects that can be configured in a disk group is limited by the size of the private region which stores configuration information about every object in the disk group. Each disk in the disk group that has a private region stores a separate copy of this configuration database.

See "Creating a disk group" on page 170.

Checking disk group configuration copies and logs (vxse_dg2)

To check whether a disk group has too many or too few disk group configuration copies, and whether a disk group has too many or too few copies of the disk group log, run rule `vxse_dg2`.

Checking "on disk config" size (vxse_dg3)

To check whether a disk group has the correct "on disk config" size, run rule `vxse_dg3`.

Checking the version number of disk groups (vxse_dg4)

To check the version number of a disk group, run rule `vxse_dg4`.

For optimum results, your disk groups should have the latest version number that is supported by the installed version of VxVM.

See "Upgrading a disk group" on page 208.

Checking the number of configuration copies in a disk group (vxse_dg5)

To find out whether a disk group has only a single VxVM configured disk, run rule `vxse_dg5`.

See "Creating and administering disk groups" on page 165.

Checking for non-imported disk groups (vxse_dg6)

To check for disk groups that are visible to VxVM but not imported, run rule `vxse_dg6`.

See "Importing a disk group" on page 174.

Checking for initialized VM disks that are not in a disk group (vxse_disk)

To find out whether there are any initialized disks that are not a part of any disk group, run rule `vxse_disk`. This prints out a list of disks, indicating whether they are part of a disk group or unassociated.

See "Adding a disk to a disk group" on page 171.

Checking volume redundancy (vxse_redundancy)

To check whether a volume is redundant, run rule `vxse_redundancy`.

This rule displays a list of volumes together with the number of mirrors that are associated with each volume. If `vxse_redundancy` shows that a volume does not have an associated mirror, your data is at risk in the event of a disk failure, and you should rectify the situation by creating a mirror for the volume.

See "Adding a mirror to a volume" on page 271.

Checking states of plexes and volumes (vxse_volplex)

To check whether your disk groups contain unused objects (such as plexes and volumes), run rule `vxse_volplex`. In particular, this rule notifies you if any of the following conditions exist:

- disabled plexes
- detached plexes
- stopped volumes
- disabled volumes
- disabled logs
- failed plexes

- volumes needing recovery

See "Reattaching plexes" on page 231.

See "Starting a volume" on page 271.

See the *Veritas Volume Manager Troubleshooting Guide*.

Disk striping

Striping enables you to enhance your system's performance. Several rules enable you to monitor important parameters such as the number of columns in a stripe plex or RAID-5 plex, and the stripe unit size of the columns.

Checking the configuration of large mirrored-stripe volumes (vxse_mirstripe)

To check whether large mirror-striped volumes should be reconfigured as striped-mirror volumes, run rule `vxse_mirstripe`.

A large mirrored-striped volume should be reconfigured, using relayout, as a striped-mirror volume to improve redundancy and enhance recovery time after failure.

See "Converting between layered and non-layered volumes" on page 300.

Checking the number of columns in RAID-5 volumes (vxse_raid5)

To check whether RAID-5 volumes have too few or too many columns, run rule `vxse_raid5`.

By default, this rule assumes that a RAID-5 plex should have more than 4 columns and fewer than 8 columns.

See "Performing online relayout" on page 294.

Checking the stripe unit size of striped volumes (vxse_stripes1)

By default, rule `vxse_stripes1` reports a violation if a volume's stripe unit size is not set to an integer multiple of 8KB.

See "Performing online relayout" on page 294.

Checking the number of columns in striped volumes (vxse_stripes2)

The default values for the number of columns in a striped plex are 16 and 3. By default, rule `vxse_stripes2` reports a violation if a striped plex in your volume has fewer than 3 columns or more than 16 columns.

See "Performing online relayout" on page 294.

Disk sparing and relocation management

The hot-relocation feature of VxVM uses spare disks in a disk group to recreate volume redundancy after disk failure.

Checking the number of spare disks in a disk group (vxse_spares)

This "best practice" rule assumes that between 10% and 20% of disks in a disk group should be allocated as spare disks. By default, `vxse_spares` checks that a disk group falls within these limits.

See "Administering hot-relocation" on page 379.

Hardware failures

VxVM maintains information about failed disks and disabled controllers.

Checking for disk failures and disabled controllers (vxse_dc_failures)

Rule `vxse_dc_failures` can be used to discover if the system has any failed disks or disabled controllers.

Rootability

The root disk can be put under VxVM control and mirrored.

Checking the validity of root mirrors (vxse_rootmir)

Rule `vxse_rootmir` can be used to confirm that the root mirrors are set up correctly.

See "Rootability" on page 102.

System name

The system name that is known to VxVM can be checked for consistency.

Checking the system name (vxse_host)

Rule `vxse_host` can be used to confirm that the system name in the file `/etc/vx/volboot` is the same as the name that was assigned to the system when it was booted.

Rule definitions and attributes

You can use the `info` keyword to show a description of a rule.

See "Discovering what a rule does" on page 447.

Table 15-1 lists the available rule definitions, and rule attributes and their default values.

Table 15-1 Rule definitions in Storage Expert

Rule	Description
vxse_dc_failures	Checks and points out failed disks and disabled controllers.
vxse_dg1	Checks for disk group configurations in which the disk group has become too large.
vxse_dg2	Checks for disk group configurations in which the disk group has too many or too few disk group configuration copies, and if the disk group has too many or too few disk group log copies.
vxse_dg3	Checks disk group configurations to verify that the disk group has the correct "on disk config" size.
vxse_dg4	Checks for disk groups that do not have a current version number, and which may need to be upgraded.
vxse_dg5	Checks for disk groups in which there is only one VxVM configured disk.
vxse_dg6	Checks for disk groups that are seen, but which are not imported.
vxse_disk	Checks for disks that are initialized, but are not part of any disk group.
vxse_disklog	Checks for physical disks that have more than one RAID-5 log.
vxse_drl1	Checks for large mirror volumes that do not have an associated DRL log.
vxse_drl2	Checks for large mirror volumes that do not have DRL log that is mirrored.
vxse_host	Checks that the system "hostname" in the `/etc/vx/volboot` file matches the hostname that was assigned to the system when it was booted.
vxse_mirstripe	Checks for large mirror-striped volumes that should be striped-mirrors.
vxse_raid5	Checks for RAID-5 volumes that are too narrow or too wide.

Table 15-1 Rule definitions in Storage Expert

Rule	Description
vxse_raid5log1	Checks for RAID-5 volumes that do not have an associated log.
vxse_raid5log2	Checks for recommended minimum and maximum RAID-5 log sizes.
vxse_raid5log3	Checks for large RAID-5 volumes that do not have a mirrored RAID-5 log.
vxse_redundancy	Checks the redundancy of volumes.
vxse_rootmir	Checks that all root mirrors are set up correctly.
vxse_spares	Checks that the number of spare disks in a disk group is within the VxVM "Best Practices" thresholds.
vxse_stripes1	Checks for stripe volumes whose stripe unit is not a multiple of the default stripe unit size.
vxse_stripes2	Checks for stripe volumes that have too many or too few columns.
vxse_volplex	Checks for volumes and plexes that are in various states, such as: ■ disabled plexes ■ detached plexes ■ stopped volumes ■ disabled volumes ■ disabled logs ■ failed plexes ■ volumes needing recovery

You can use the `list` and `check` keywords to show what attributes are available for a rule and to display the default values of these attributes.

See "Running a rule" on page 447.

Table 15-2 lists the available rule attributes and their default values.

Table 15-2 Rule attributes and default attribute values

Rule	Attribute	Default value	Description
vxse_dc_failures	–	-	No user-configurable variables.
vxse_dg1	max_disks_per_dg	250	Maximum number of disks in a disk group. Warn if a disk group has more disks than this.
vxse_dg2	–	-	No user-configurable variables.
vxse_dg3	–	-	No user-configurable variables.
vxse_dg4	–	-	No user-configurable variables.
vxse_dg5	–	-	No user-configurable variables.
vxse_dg6	–	-	No user-configurable variables.
vxse_disk	-	-	No user-configurable variables.
vxse_disklog	-	-	No user-configurable variables.
vxse_drl1	mirror_threshold	1g (1GB)	Large mirror threshold size. Warn if a mirror is larger than this and does not have an attached DRL log.
vxse_drl2	large_mirror_size	20g (20GB)	Large mirror-stripe threshold size. Warn if a mirror-stripe volume is larger than this.
vxse_host	-	-	No user-configurable variables.

Table 15-2 Rule attributes and default attribute values

Rule	Attribute	Default value	Description
vxse_mirstripe	large_mirror_size	1g (1GB)	Large mirror-stripe threshold size. Warn if a mirror-stripe volume is larger than this.
	nsd_threshold	8	Large mirror-stripe number of subdisks threshold. Warn if a mirror-stripe volume has more subdisks than this.
vxse_raid5	too_narrow_raid5	4	Minimum number of RAID-5 columns. Warn if actual number of RAID-5 columns is less than this.
	too_wide_raid5	8	Maximum number of RAID-5 columns. Warn if the actual number of RAID-5 columns is greater than this.
vxse_raid5log1	-	-	No user-configurable variables.
vxse_raid5log2	r5_max_size	1g (1GB)	Maximum RAID-5 log check size. Warn if a RAID-5 log is larger than this.
	r5_min_size	64m (64MB)	Minimum RAID-5 log check size. Warn if a RAID-5 log is smaller than this.
vxse_raid5log3	large_vol_size	20g (20GB)	Large RAID-5 volume threshold size. Warn if a RAID-5 volume with a non-mirrored RAID-5 log is larger than this.

Table 15-2 Rule attributes and default attribute values

Rule	Attribute	Default value	Description
vxse_redundancy	volume_redundancy	0	Volume redundancy check. The value of 2 performs a mirror redundancy check. A value of 1 performs a RAID-5 redundancy check. The default value of 0 performs no redundancy check.
vxse_rootmir	-	-	No user-configurable variables.
vxse_spares	max_disk_spare_ratio	20	Maximum percentage of spare disks in a disk group. Warn if the percentage of spare disks is greater than this.
	min_disk_spare_ratio	10	Minimum percentage of spare disks in a disk group. Warn if the percentage of spare disks is less than this.
vxse_stripes1	default _stripeunit	8k (8KB)	Stripe unit size for stripe volumes. Warn if a stripe does not have a stripe unit which is an integer multiple of this value.
vxse_stripes2	too_narrow_stripe	3	Minimum number of columns in a striped plex. Warn if a striped volume has fewer columns than this.
	too_wide_stripe	16	Maximum number of columns in a striped plex. Warn if a striped volume has more columns than this.

Table 15-2 Rule attributes and default attribute values

Rule	Attribute	Default value	Description
vxse_volplex	-	-	No user-configurable variables.

Chapter 16

Performance monitoring and tuning

Veritas Volume Manager (VxVM) can improve overall system performance by optimizing the layout of data storage on the available hardware. This chapter contains guidelines establishing performance priorities, for monitoring performance, and for configuring your system appropriately.

Performance guidelines

VxVM allows you to optimize data storage performance using the following two strategies:

- Balance the I/O load among the available disk drives.
- Use striping and mirroring to increase I/O bandwidth to the most frequently accessed data.

VxVM also provides data redundancy (through mirroring and RAID-5) that allows continuous access to data in the event of disk failure.

Data assignment

When deciding where to locate file systems, you, as a system administrator, typically attempt to balance I/O load among the available disk drives. The effectiveness of this approach is limited by the difficulty of anticipating future usage patterns, as well as the inability to split file systems across the drives. For example, if a single file system receives the most disk accesses, moving the file system to another drive also moves the bottleneck to that drive.

VxVM can split volumes across multiple drives. This permits you a finer level of granularity when locating data. After measuring actual access patterns, you can adjust your previous decisions on the placement of file systems. You can reconfigure volumes online without adversely impacting their availability.

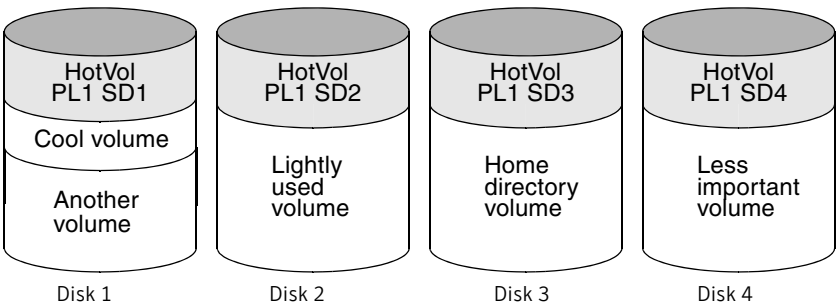
Striping

Striping improves access performance by cutting data into slices and storing it on multiple devices that can be accessed in parallel. Striped plexes improve access performance for both read and write operations.

Having identified the most heavily accessed volumes (containing file systems or databases), you can increase access bandwidth to this data by striping it across portions of multiple disks.

Figure 16-1 shows an example of a single volume (`HotVol`) that has been identified as a data-access bottleneck. This volume is striped across four disks, leaving the remaining space on these disks free for use by less-heavily used volumes.

Figure 16-1 Use of striping for optimal data access



Mirroring

Note: You need a full license to use this feature.

Mirroring stores multiple copies of data on a system. When properly applied, mirroring provides continuous availability of data and protection against data loss due to physical media failure. Mirroring improves the chance of data recovery in the event of a system crash or the failure of a disk or other hardware.

In some cases, you can also use mirroring to improve I/O performance. Unlike striping, the performance gain depends on the ratio of reads to writes in the disk accesses. If the system workload is primarily write-intensive (for example, greater than 30 percent writes), mirroring can result in reduced performance.

Combining mirroring and striping

Note: You need a full license to use this feature.

Mirroring and striping can be used together to achieve a significant improvement in performance when there are multiple I/O streams.

Striping provides better throughput because parallel I/O streams can operate concurrently on separate devices. Serial access is optimized when I/O exactly fits across all stripe units in one stripe.

Because mirroring is generally used to protect against loss of data due to disk failures, it is often applied to write-intensive workloads which degrades throughput. In such cases, combining mirroring with striping delivers both high availability and increased throughput.

A mirrored-stripe volume may be created by striping half of the available disks to form one striped data plex, and striping the remaining disks to form the other striped data plex in the mirror. This is often the best way to configure a set of disks for optimal performance with reasonable reliability. However, the failure of a single disk in one of the plexes makes the entire plex unavailable.

Alternatively, you can arrange equal numbers of disks into separate mirror volumes, and then create a striped plex across these mirror volumes to form a striped-mirror volume (see “Mirroring plus striping (striped-mirror, RAID-1+0 or RAID-10)” on page 43). The failure of a single disk in a mirror does not take the disks in the other mirrors out of use. A striped-mirror layout is preferred over a mirrored-stripe layout for large volumes or large numbers of disks.

RAID-5

Note: You need a full license to use this feature.

RAID-5 offers many of the advantages of combined mirroring and striping, but requires less disk space. RAID-5 read performance is similar to that of striping and RAID-5 parity offers redundancy similar to mirroring. Disadvantages of RAID-5 include relatively slow write performance.

RAID-5 is not usually seen as a way of improving throughput performance except in cases where the access patterns of applications show a high ratio of reads to writes.

Volume read policies

To help optimize performance for different types of volumes, VxVM supports the following read policies on data plexes:

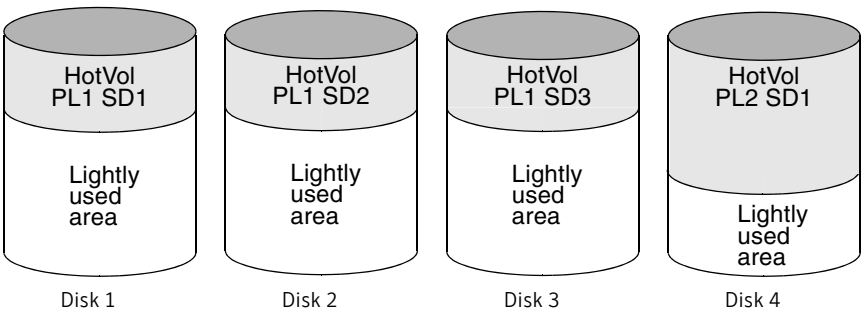
- `round`—a *round-robin* read policy, where all plexes in the volume take turns satisfying read requests to the volume.
- `prefer`—a *preferred-plex* read policy, where the plex with the highest performance usually satisfies read requests. If that plex fails, another plex is accessed.
- `select`—default read policy, where the appropriate read policy for the configuration is selected automatically. For example, `prefer` is selected when there is only one striped plex associated with the volume, and `round` is selected in most other cases.

Note: You cannot set the read policy on a RAID-5 data plex. RAID-5 plexes have their own read policy (RAID).

For instructions on how to configure the read policy for a volume's data plexes, see "Changing the read policy for mirrored volumes" on page 289.

In the configuration example shown in Figure 16-2, the read policy of the mirrored-stripe volume labeled `Hot Vol` is set to `prefer` for the striped plex `PL1`. This policy distributes the load when reading across the otherwise lightly-used disks in `PL1`, as opposed to the single disk in plex `PL2`. (`HotVol` is an example of a mirrored-stripe volume in which one data plex is striped and the other data plex is concatenated.)

Figure 16-2 Use of mirroring and striping for improved performance



Note: To improve performance for read-intensive workloads, you can attach up to 32 data plexes to the same volume. However, this would usually be an ineffective use of disk space for the gain in read performance.

Performance monitoring

As a system administrator, you have two sets of priorities for setting priorities for performance. One set is *physical*, concerned with hardware such as disks and controllers. The other set is *logical*, concerned with managing software and its operation.

Setting performance priorities

The important physical performance characteristics of disk hardware are the relative amounts of I/O on each drive, and the concentration of the I/O within a drive to minimize seek time. Based on monitored results, you can then move the location of subdisks to balance I/O activity across the disks.

The logical priorities involve software operations and how they are managed. Based on monitoring, you may choose to change the layout of certain volumes to improve their performance. You might even choose to reduce overall throughput to improve the performance of certain critical volumes. Only you can decide what is important on your system and what trade-offs you need to make.

Best performance is usually achieved by striping and mirroring all volumes across a reasonable number of disks and mirroring between controllers, when possible. This procedure tends to even out the load between all disks, but it can make VxVM more difficult to administer. For large numbers of disks (hundreds or thousands), set up disk groups containing 10 disks, where each group is used to create a striped-mirror volume. This technique provides good performance while easing the task of administration.

Obtaining performance data

VxVM provides two types of performance information: I/O statistics and I/O traces. Each of these can help in performance monitoring. You can obtain I/O statistics using the `vxstat` command, and I/O traces using the `vxtrace` command. A brief discussion of each of these utilities may be found in the following sections.

Tracing volume operations

Use the `vxtrace` command to trace operations on specified volumes, kernel I/O object types or devices. The `vxtrace` command either prints kernel I/O errors or I/O trace records to the standard output or writes the records to a file in binary format. Binary trace records written to a file can also be read back and formatted by `vxtrace`.

If you do not specify any operands, `vxtrace` reports either all error trace data or all I/O trace data on all virtual disk devices. With error trace data, you can select all accumulated error trace data, wait for new error trace data, or both of these (this is the default action). Selection can be limited to a specific disk group, to specific VxVM kernel I/O object types, or to particular named objects or devices.

For detailed information about how to use `vxtrace`, refer to the `vxtrace`(1M) manual page.

Printing volume statistics

Use the `vxstat` command to access information about activity on volumes, plexes, subdisks, and disks under VxVM control, and to print summary statistics to the standard output. These statistics represent VxVM activity from the time the system initially booted or from the last time the counters were reset to zero. If no VxVM object name is specified, statistics from all volumes in the configuration database are reported.

VxVM records the following I/O statistics:

- count of operations
- number of blocks transferred (one operation can involve more than one block)
- average operation time (which reflects the total time through the VxVM interface and is not suitable for comparison against other statistics programs)

These statistics are recorded for logical I/O including reads, writes, atomic copies, verified reads, verified writes, plex reads, and plex writes for each volume. As a result, one write to a two-plex volume results in at least five operations: one for each plex, one for each subdisk, and one for the volume. Also, one read that spans two subdisks shows at least four reads—one read for each subdisk, one for the plex, and one for the volume.

VxVM also maintains other statistical data. For each plex, it records read and write failures. For volumes, it records corrected read and write failures in addition to read and write failures.

To reset the statistics information to zero, use the `-r` option. This can be done for all objects or for only those objects that are specified. Resetting just prior to

an operation makes it possible to measure the impact of that particular operation.

The following is an example of output produced using the `vxstat` command:

```
                 OPERATIONS          BLOCKS          AVG TIME(ms)
TYP  NAME       READ   WRITE     READ     WRITE     READ     WRITE
vol  blop          0       0        0         0      0.0       0.0
vol  foobarvol     0       0        0         0      0.0       0.0
vol  rootvol   73017  181735   718528   1114227     26.8      27.9
vol  swapvol   13197   20252   105569    162009     25.8     397.0
vol  testvol       0       0        0         0      0.0       0.0
```

Additional volume statistics are available for RAID-5 configurations.

For detailed information about how to use `vxstat`, refer to the `vxstat`(1M) manual page.

Using performance data

When you have gathered performance data, you can use it to determine how to configure your system to use resources most effectively. The following sections provide an overview of how you can use this data.

Using I/O statistics

Examination of the I/O statistics can suggest how to reconfigure your system. You should examine two primary statistics: volume I/O activity and disk I/O activity.

Before obtaining statistics, reset the counters for all existing statistics using the `vxstat -r` command. This eliminates any differences between volumes or disks due to volumes being created, and also removes statistics from boot time (which are not usually of interest).

After resetting the counters, allow the system to run during typical system activity. Run the application or workload of interest on the system to measure its effect. When monitoring a system that is used for multiple purposes, try not to exercise any one application more than usual. When monitoring a time-sharing system with many users, let statistics accumulate for several hours during the normal working day.

To display volume statistics, enter the `vxstat` command with no arguments. The following is a typical display of volume statistics:

```
                   OPERATIONS          BLOCKS          AVG TIME(ms)
TYP  NAME        READ   WRITE     READ     WRITE     READ     WRITE
vol  archive      865     807     5722      3809     32.5      24.0
vol  home        2980    5287     6504     10550     37.7     221.1
vol  local      49477   49230   507892    204975     28.5      33.5
vol  rootvol   102906  342664  1085520   1962946     28.1      25.6
vol  src        79174   23603   425472    139302     22.4      30.9
vol  swapvol    22751   32364   182001    258905     25.3     323.2
```

Such output helps to identify volumes with an unusually large number of operations or excessive read or write times.

To display disk statistics, use the `vxstat -d` command. The following is a typical display of disk statistics:

```
                    OPERATIONS          BLOCKS          AVG TIME(ms)
TYP  NAME      READ    WRITE     READ     WRITE      READ   WRITE
dm   mydg01   40473   174045   455898    951379     29.5    35.4
dm   mydg02   32668    16873   470337    351351     35.2   102.9
dm   mydg03   55249    60043   780779    731979     35.3    61.2
dm   mydg04   11909    13745   114508    128605     25.0    30.7
```

If you need to move the volume named `archive` onto another disk, use the following command to identify on which disks it lies:

```
# vxprint -g mydg -tvh archive
```

The following is an extract from typical output:

```
V   NAME        RVG/VSET/CO  KSTATE   STATE    LENGTH     READPOL    REFPLEX    UTYPE
PL  NAME        VOLUME       KSTATE   STATE    LENGTH     LAYOUT     NCOL/WDTH  MODE
SD  NAME        PLEX         DISK     DISKOFFS LENGTH     [COL/]OFF  DEVICE     MODE

v   archive     -            ENABLED  ACTIVE   20480      SELECT     -          fsgen
pl  archive-01  archive      ENABLED  ACTIVE   20480      CONCAT     -          RW
sd  mydg03-03   archive-01   mydg03   0        40960      0          c1t2d0     ENA
```

Note: Your system may use device names that differ from these examples. For more information on device names, see "Administering disks" on page 77.

The subdisks line (beginning `sd`) indicates that the volume `archive` is on disk `mydg03`. To move the volume off `mydg03`, use the following command:

```
# vxassist -g mydg move archive !mydg03 dest_disk
```

Here *dest_disk* is the destination disk to which you want to move the volume. It is not necessary to specify a destination disk. If you do not specify a destination disk, the volume is moved to an available disk with enough space to contain the volume.

For example, to move a volume from disk `mydg03` to disk `mydg04`, in the disk group, `mydg`, use the following command:

```
# vxassist -g mydg move archive !mydg03 mydg04
```

This command indicates that the volume is to be reorganized so that no part of it remains on `mydg03`.

Note: The Veritas Enterprise Administrator (VEA) has a graphical user interface (GUI), which provides an easier way to move pieces of volumes between disks. You may find that approach preferable to using the command line.

If two volumes (other than the root volume) on the same disk are busy, move them so that each is on a different disk.

If one volume is particularly busy (especially if it has unusually large average read or write times), stripe the volume (or split the volume into multiple pieces, with each piece on a different disk). If done online, converting a volume to use striping requires sufficient free space to store an extra copy of the volume. If sufficient free space is not available, a backup copy can be made instead. To convert a volume, create a striped plex as a mirror of the volume and then remove the old plex. For example, the following commands stripe the volume `archive` across disks `mydg02`, `mydg03`, and `mydg04` in the disk group, `mydg`, and then remove the original plex `archive-01`:

```
# vxassist -g mydg mirror archive layout=stripe mydg02 mydg03 \
  mydg04
# vxplex -g mydg -o rm dis archive-01
```

After reorganizing any particularly busy volumes, check the disk statistics. If some volumes have been reorganized, clear statistics first and then accumulate statistics for a reasonable period of time.

If some disks appear to be excessively busy (or have particularly long read or write times), you may want to reconfigure some volumes. If there are two relatively busy volumes on a disk, move them closer together to reduce seek times on the disk. If there are too many relatively busy volumes on one disk, move them to a disk that is less busy.

Use I/O tracing (or subdisk statistics) to determine whether volumes have excessive activity in particular regions of the volume. If the active regions can be identified, split the subdisks in the volume and move those regions to a less busy disk.

Caution: Striping a volume, or splitting a volume across multiple disks, increases the chance that a disk failure results in failure of that volume. For example, if five volumes are striped across the same five disks, then failure of any one of the five disks requires that all five volumes be restored from a backup. If each volume were on a separate disk, only one volume would need to be restored. Use mirroring or RAID-5 to reduce the chance that a single disk failure results in failure of a large number of volumes.

Note that file systems and databases typically shift their use of allocated space over time, so this position-specific information on a volume is often not useful. Databases are reasonable candidates for moving to non-busy disks if the space used by a particularly busy index or table can be identified.

Examining the ratio of reads to writes helps to identify volumes that can be mirrored to improve their performance. If the read-to-write ratio is high, mirroring can increase performance as well as reliability. The ratio of reads to

writes where mirroring can improve performance depends greatly on the disks, the disk controller, whether multiple controllers can be used, and the speed of the system bus. If a particularly busy volume has a high ratio of reads to writes, it is likely that mirroring can significantly improve performance of that volume.

Using I/O tracing

I/O statistics provide the data for basic performance analysis; I/O traces serve for more detailed analysis. With an I/O trace, focus is narrowed to obtain an event trace for a specific workload. This helps to explicitly identify the location and size of a hot spot, as well as which application is causing it.

Using data from I/O traces, real work loads on disks can be simulated and the results traced. By using these statistics, you can anticipate system limitations and plan for additional resources.

For information on using the `vxdmpadm` command to gather I/O statistics for a DMP node, path, or enclosure, see "Gathering and displaying I/O statistics" on page 144. You can also use the `vxdmpadm` command to change the I/O load-balancing policy for an enclosure as described in "Specifying the I/O policy" on page 147.

Tuning VxVM

This section describes how to adjust the tunable parameters that control the system resources used by VxVM. Depending on the system resources that are available, adjustments may be required to the values of some tunable parameters to optimize performance.

General tuning guidelines

VxVM is optimally tuned for most configurations ranging from small systems to larger servers. In cases where tuning can be used to increase performance on larger systems at the expense of a valuable resource (such as memory), VxVM is generally tuned to run on the smallest supported configuration. Any tuning changes must be performed with care, as they may adversely affect overall system performance or may even leave VxVM unusable.

Various mechanisms exist for tuning VxVM. Many parameters can be tuned using the System Management Homepage (SMH) or the `kctune` utility. Other values can only be tuned using the command line interface to VxVM.

Tuning guidelines for large systems

On smaller systems (with fewer than a hundred disk drives), tuning is unnecessary and VxVM is capable of adopting reasonable defaults for all configuration parameters. On larger systems, configurations can require additional control over the tuning of these parameters, both for capacity and performance reasons.

Generally, only a few significant decisions must be made when setting up VxVM on a large system. One is to decide on the size of the disk groups and the number of configuration copies to maintain for each disk group. Another is to choose the size of the private region for all the disks in a disk group.

Larger disk groups have the advantage of providing a larger free-space pool for the `vxassist`(1M) command to select from, and also allow for the creation of larger volumes. Smaller disk groups do not require as large a configuration database and so can exist with smaller private regions. Very large disk groups can eventually exhaust the private region size in the disk group with the result that no more configuration objects can be added to that disk group. At that point, the configuration either has to be split into multiple disk groups, or the private regions have to be enlarged. This involves re-initializing each disk in the disk group (and can involve reconfiguring everything and restoring from backup).

A general recommendation for users of disk array subsystems is to create a single disk group for each array so the disk group can be physically moved as a unit between systems.

Number of configuration copies for a disk group

Selection of the number of configuration copies for a disk group is based on a trade-off between redundancy and performance. As a general rule, reducing the number configuration copies in a disk group speeds up initial access of the disk group, initial startup of the `vxconfigd` daemon, and transactions performed within the disk group. However, reducing the number of configuration copies also increases the risk of complete loss of the configuration database, which results in the loss of all objects in the database and of all data in the disk group.

The default policy for configuration copies in the disk group is to allocate a configuration copy for each controller identified in the disk group, or for each target that contains multiple addressable disks. This provides a sufficient degree of redundancy, but can lead to a large number of configuration copies under some circumstances. If this is the case, we recommended that you limit the number of configuration copies to a maximum of 4. Distribute the copies across separate controllers or targets to enhance the effectiveness of this redundancy.

To set the number of configuration copies for a new disk group, use the `nconfig` operand with the `vxdg init` command (see the `vxdg`(1M) manual page for details).

You can also change the number of copies for an existing group by using the `vxedit set` command (see the `vxedit`(1M) manual page). For example, to configure five configuration copies for the disk group, `bigdg`, use the following command:

```
# vxedit set nconfig=5 bigdg
```

Changing the values of tunables

Tunables are modified by using the System Management Homepage (SMH) or the `kctune` utility. Changed tunables take effect only after relinking the kernel and booting the system from the new kernel.

The values of system tunables can be examined or changed in SMH by selecting **Home > System Configuration > Kernel Configuration > Tunables**.

Most DMP tunables may be set online (without requiring a reboot) by using the `vxdmpadm` command as shown here:

```
# vxdmpadm settune dmp_tunable=value
```

The values of these tunables can be displayed by using this command:

```
# vxdmpadm gettune [dmp_tunable]
```

The `vxdmpadm` command also allows you to configure how DMP responds to I/O errors at the level of the paths to individual arrays. See “Administering DMP using vxdmpadm” on page 139 for details.

Tunable parameters

The following sections describe specific tunable parameters.

dmp_cache_open

If set to `on`, the first open of a device that is performed by an array support library (ASL) is cached. This enhances the performance of device discovery by minimizing the overhead caused by subsequent opens by ASLs. If set to `off`, caching is not performed. The default value is `off`.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_daemon_count

The number of kernel threads that are available for servicing path error handling, path restoration and other DMP administrative tasks.

The default number of threads is 10.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_delayq_interval

How long DMP should wait before retrying I/O after an array fails over to a standby path. Some disk arrays are not be capable of accepting I/O requests immediately after failover.

The default value is 15 seconds.

The value of this tunable is changed by using the `vxdmpadm settune` command.

"Configuring DMP path restoration policies" on page 160dmp_failed_io_threshold

The time limit that DMP waits for a failed I/O request to return before the device is marked as INSANE, I/O is avoided on the path, and any remaining failed I/O requests are returned to the application layer without performing any error analysis.

The default value is 57600 seconds (16 hours).

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_fast_recovery

Whether DMP should attempt to obtain SCSI error information directly from the HBA interface. Setting the value to on can potentially provide faster error recovery, provided that the HBA interface supports the error enquiry feature.

If set to `off`, the HBA interface is not used. This is the default setting.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_health_time

DMP detects intermittently failing paths, and prevents I/O requests from being sent on them. The value of `dmp_health_time` represents the time in seconds for which a path must stay healthy. If a path's state changes back from enabled to disabled within this time period, DMP marks the path as intermittently failing, and does not re-enable the path for I/O until `dmp_path_age` seconds elapse. The default value of `dmp_health_time` is 60 seconds. A value of 0 prevents DMP from detecting intermittently failing paths.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_log_level

The level of detail that is displayed for DMP console messages. The following level values are defined:

1 Display all DMP log messages that existed in releases before 5.0. This is the default setting.

2 Display level 1 messages plus messages that relate to I/O throttling, suspected paths, repeated path failures and DMP node migration.

3 Display level 1 and 2 messages plus messages that relate to I/O errors, I/O error analysis and path media errors.

4 Display level 1, 2 and 3 messages plus messages that relate to setting or changing attributes on a path.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_path_age

The time for which an intermittently failing path needs to be monitored as healthy before DMP once again attempts to schedule I/O requests on it. The default value is 300 seconds. A value of 0 prevents DMP from detecting intermittently failing paths.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_pathswitch_blks_shift

The default number of contiguous I/O blocks (expressed as the integer exponent of a power of 2; for example 10 represents 1024 blocks) that are sent along a DMP path to an Active/Active array before switching to the next available path.

The default value of this parameter is set to 10 so that 1024 blocks (1MB) of contiguous I/O are sent over a DMP path before switching. For intelligent disk arrays with internal data caches, better throughput may be obtained by

increasing the value of this tunable. For example, for the HDS 9960 A/A array, the optimal value is between 14 and 16 for an I/O activity pattern that consists mostly of sequential reads or writes.

Note: This parameter only affects the behavior of the `balanced` I/O policy. A value of 0 disables multipathing for the policy unless the `vxdmpadm` command is used to specify a different partition size for an array as described in "Specifying the I/O policy" on page 147.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_probe_idle_lun

If DMP statistics gathering is enabled, set to `on` (default) to have the DMP path restoration thread probe idle LUNs, or to `off` to turn off this feature. (Idle LUNs are VM disks on which no I/O requests are scheduled.) The value of this tunable is only interpreted when DMP statistics gathering is enabled. Turning off statistics gathering also disables idle LUN probing.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_queue_depth

The maximum number of queued I/O requests on a path during I/O throttling. The default value is 40.

The value of this tunable is changed by using the `vxdmpadm settune` command.

See "Configuring the I/O throttling mechanism" on page 157.

dmp_restore_cycles

If the DMP restore policy is CHECK_PERIODIC, the number of cycles after which the CHECK_ALL policy is called.

The value of this tunable is only changeable by using the `vxdmpadm start restore` command.

dmp_restore_interval

The time in seconds between two invocations of the DMP path restoration thread.

The value of this tunable is only changeable by using the `vxdmpadm start restore` command.

dmp_restore_policy

The DMP restore policy, which can be set to 0 (CHECK_ALL), 1 (CHECK_DISABLED), 2 (CHECK_PERIODIC), or 3 (CHECK_ALTERNATE).

The value of this tunable is only changeable by using the `vxdmpadm start restore` command.

dmp_retry_count

If an inquiry succeeds on a path, but there is an I/O error, the number of retries to attempt on the path. The default number of retries is 30.

The value of this tunable is changed by an entry in the `/kernel/drv/vxdmp.conf` file, or by using the `vxdmpadm settune` command.

See "Configuring the response to I/O failures" on page 156.

dmp_retry_timeout

The maximum time period for which DMP retries the SCSI-3 Persistent Reserve operation with A/P arrays.

The default value is 0 seconds, which disables this feature.

This parameter has no direct effect on I/O processing by DMP.

Disabling a switch port can trigger a fabric reconfiguration, which can take time to stabilize. During this period, attempting to register PGR keys through the secondary path to an array may fail with an error condition, such as unit attention or device reset, or the return of vendor-specific sense data.

The retry period allows a fabric reconfiguration (usually a transient condition) to not be seen as an error by DMP.

Do not set the value of the retry period too high. This can delay the failover process, and result in I/O sluggishness or suppression of I/O activity during the retry period.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_scsi_timeout

Determines the timeout value to be set for any SCSI command that is sent via DMP. If the HBA does not receive a response for a SCSI command that it has sent to the device within the timeout period, the SCSI command is returned with a failure error code.

The default value is 60 seconds.

The value of this tunable is changed by using the `vxdmpadm settune` command.

dmp_stat_interval

The time interval between gathering DMP statistics. The default and minimum value is 1 second.

The value of this tunable is changed by using the `vxdmpadm settune` command.

vol_checkpt_default

The interval at which utilities performing recoveries or resynchronization operations load the current offset into the kernel as a checkpoint. A system failure during such operations does not require a full recovery, but can continue from the last reached checkpoint.

The default value of the checkpoint is 10240 sectors (10MB).

Increasing this size reduces the overhead of checkpointing on recovery operations at the expense of additional recovery following a system failure during a recovery.

vol_default_iodelay

The count in clock ticks for which utilities pause if they have been directed to reduce the frequency of issuing I/O requests, but have not been given a specific delay time. This tunable is used by utilities performing operations such as resynchronizing mirrors or rebuilding RAID-5 columns.

The default for this tunable is 50 ticks.

Increasing this value results in slower recovery operations and consequently lower system impact while recoveries are being performed.

vol_fmr_logsz

The maximum size in kilobytes of the bitmap that Non-Persistent FastResync uses to track changed blocks in a volume. The number of blocks in a volume that are mapped to each bit in the bitmap depends on the size of the volume, and this value changes if the size of the volume is changed. For example, if the volume size is 1 gigabyte and the system block size is 1024 bytes, a `vol_fmr_logsz` value of 4 yields a map contains 32,768 bits, each bit representing one region of 32 blocks.

The larger the bitmap size, the fewer the number of blocks that are mapped to each bit. This can reduce the amount of reading and writing required on resynchronization, at the expense of requiring more non-pageable kernel memory for the bitmap. Additionally, on clustered systems, a larger bitmap size increases the latency in I/O performance, and it also increases the load on the private network between the cluster members. This is because every other member of the cluster must be informed each time a bit in the map is marked.

Since the region size must be the same on all nodes in a cluster for a shared volume, the value of the `vol_fmr_logsz` tunable on the master node overrides the tunable values on the slave nodes, if these values are different. Because the value of a shared volume can change, the value of `vol_fmr_logsz` is retained for the life of the volume.

In configurations which have thousands of mirrors with attached snapshot plexes, the total memory overhead can represent a significantly higher overhead in memory consumption than is usual for VxVM.

The default value of this tunable is 4KB. The maximum and minimum permitted values are 1KB and 8KB.

Note: The value of this tunable does not have any effect on Persistent FastResync.

vol_max_vol

The maximum number of volumes that can be created on the system. This value can be set to between 1 and the maximum number of minor numbers representable in the system.

The default value for this tunable is 16777215.

vol_maxio

The maximum size of logical I/O operations that can be performed without breaking up the request. I/O requests to VxVM that are larger than this value are broken up and performed synchronously. Physical I/O requests are broken up based on the capabilities of the disk device and are unaffected by changes to this maximum logical request limit.

The default value for this tunable is 256 sectors (256KB).

Note: The value of `voliomem_maxpool_sz` must be at least 10 times greater than the value of `vol_maxio`.

If DRL sequential logging is configured, the value of `voldrl_min_regionsz` must be set to at least half the value of `vol_maxio`.

vol_maxioctl

The maximum size of data that can be passed into VxVM via an `ioctl` call. Increasing this limit allows larger operations to be performed. Decreasing the limit is not generally recommended, because some utilities depend upon

performing operations of a certain size and can fail unexpectedly if they issue oversized `ioctl` requests.

The default value for this tunable is 32768 bytes (32KB).

vol_maxparallelio

The number of I/O operations that the `vxconfigd`(1M) daemon is permitted to request from the kernel in a single `VOL_VOLDIO_READ` per `VOL_VOLDIO_WRITE` `ioctl` call.

The default value for this tunable is 256. It is not desirable to change this value.

vol_maxspecialio

The maximum size of an I/O request that can be issued by an `ioctl` call. Although the `ioctl` request itself can be small, it can request a large I/O request be performed. This tunable limits the size of these I/O requests. If necessary, a request that exceeds this value can be failed, or the request can be broken up and performed synchronously.

The default value for this tunable is 256 sectors (256KB).

Raising this limit can cause difficulties if the size of an I/O request causes the process to take more memory or kernel virtual mapping space than exists and thus deadlock. The maximum limit for `vol_maxspecialio` is 20% of the smaller of physical memory or kernel virtual memory. It is inadvisable to go over this limit, because deadlock is likely to occur.

If stripes are larger than `vol_maxspecialio`, full stripe I/O requests are broken up, which prevents full-stripe read/writes. This throttles the volume I/O throughput for sequential I/O or larger I/O requests.

This tunable limits the size of an I/O request at a higher level in VxVM than the level of an individual disk. For example, for an 8 by 64KB stripe, a value of 256KB only allows I/O requests that use half the disks in the stripe; thus, it cuts potential throughput in half. If you have more columns or you have used a larger interleave factor, then your relative performance is worse.

This tunable must be set, as a minimum, to the size of your largest stripe (RAID-0 or RAID-5).

vol_subdisk_num

The maximum number of subdisks that can be attached to a single plex. There is no theoretical limit to this number, but it has been limited to a default value of 4096. This default can be changed, if required.

volcvm_smartsync

If set to `0`, `volcvm_smartsync` disables SmartSync on shared disk groups. If set to `1`, this parameter enables the use of SmartSync with shared disk groups. See"SmartSync recovery accelerator" on page 62 for more information.

voldrl_max_drtregs

The maximum number of dirty regions that can exist on the system for non-sequential DRL on volumes. A larger value may result in improved system performance at the expense of recovery time. This tunable can be used to regulate the worse-case recovery time for the system following a failure.

The default value for this tunable is 2048.

voldrl_max_seq_dirty

The maximum number of dirty regions allowed for sequential DRL. This is useful for volumes that are usually written to sequentially, such as database logs. Limiting the number of dirty regions allows for faster recovery if a crash occurs.

The default value for this tunable is 3.

voldrl_min_regionsz

The minimum number of sectors for a dirty region logging (DRL) volume region. With DRL, VxVM logically divides a volume into a set of consecutive regions. Larger region sizes tend to cause the cache hit-ratio for regions to improve. This improves the write performance, but it also prolongs the recovery time.

The VxVM kernel currently sets the default value for this tunable to 512 sectors.

Note: If DRL sequential logging is configured, the value of `voldrl_min_regionsz` must be set to at least half the value of `vol_maxio`.

voliomem_chunk_size

The granularity of memory chunks used by VxVM when allocating or releasing system memory. A larger granularity reduces CPU overhead due to memory allocation by allowing VxVM to retain hold of a larger amount of memory.

The default size for this tunable is 64KB.

voliomem_maxpool_sz

The maximum memory requested from the system by VxVM for internal purposes. This tunable has a direct impact on the performance of VxVM as it prevents one I/O operation from using all the memory in the system.

VxVM allocates two pools that can grow up to `voliomem_maxpool_sz`, one for RAID-5 and one for mirrored volumes.

A write request to a RAID-5 volume that is greater than `voliomem_maxpool_sz/10` is broken up and performed in chunks of size `voliomem_maxpool_sz/10`.

A write request to a mirrored volume that is greater than `voliomem_maxpool_sz/2` is broken up and performed in chunks of size `voliomem_maxpool_sz/2`.

The default value for this tunable is 4M.

Note: The value of `voliomem_maxpool_sz` must be at least 10 times greater than the value of `vol_maxio`.

voliot_errbuf_dflt

The default size of the buffer maintained for error tracing events. This buffer is allocated at driver load time and is not adjustable for size while VxVM is running.

The default size for this buffer is 16384 bytes (16KB).

Increasing this buffer can provide storage for more error events at the expense of system memory. Decreasing the size of the buffer can result in an error not being detected via the tracing device. Applications that depend on error tracing to perform some responsive action are dependent on this buffer.

voliot_iobuf_default

The default size for the creation of a tracing buffer in the absence of any other specification of desired kernel buffer size as part of the trace `ioctl`.

The default size of this tunable is 8192 bytes (8KB).

If trace data is often being lost due to this buffer size being too small, then this value can be tuned to a more generous amount.

voliot_iobuf_limit

The upper limit to the size of memory that can be used for storing tracing buffers in the kernel. Tracing buffers are used by the VxVM kernel to store the

tracing event records. As trace buffers are requested to be stored in the kernel, the memory for them is drawn from this pool.

Increasing this size can allow additional tracing to be performed at the expense of system memory usage. Setting this value to a size greater than can readily be accommodated on the system is inadvisable.

The default value for this tunable is 131072 bytes (128KB).

voliot_iobuf_max

The maximum buffer size that can be used for a single trace buffer. Requests of a buffer larger than this size are silently truncated to this size. A request for a maximal buffer size from the tracing interface results (subject to limits of usage) in a buffer of this size.

The default size for this buffer is 65536 bytes (64KB).

Increasing this buffer can provide for larger traces to be taken without loss for very heavily used volumes. Care should be taken not to increase this value above the value for the `voliot_iobuf_limit` tunable value.

voliot_max_open

The maximum number of tracing channels that can be open simultaneously. Tracing channels are clone entry points into the tracing device driver. Each `vxtrace` process running on a system consumes a single trace channel.

The default number of channels is 32. The allocation of each channel takes up approximately 20 bytes even when not in use.

volpagemod_max_memsz

The amount of memory, measured in kilobytes, that is allocated for caching FastResync and cache object metadata. This tunable has a default value of 6144KB (6MB) of physical memory.

Note: The memory allocated for this cache is exclusively dedicated to it. It is not available for other processes or applications.

Setting the value of `volpagemod_max_memsz` below 512KB fails if cache objects or volumes that have been prepared for instant snapshot operations are present on the system.

If you do not use the FastResync or DRL features that are implemented using a version 20 DCO volume, the value of `volpagemod_max_memsz` can be set to 0. However, if you subsequently decide to enable these features, you can use the `vxtune` command to change the value to a more appropriate one:

```
# vxtune volpagemod_max_memsz value
```

where the new value is specified in kilobytes. Using the `vxtune` command to adjust the value of `volpagemod_max_memsz` does not persist across system reboots unless you also adjust the value that is configured in the `/stand/system` file.

volraid_minpool_sz

The initial amount of memory that is requested from the system by VxVM for RAID-5 operations. The maximum size of this memory pool is limited by the value of `voliomem_maxpool_sz`.

The default value for this tunable is 16348 sectors (16MB).

volraid_rsrtransmax

The maximum number of transient reconstruct operations that can be performed in parallel for RAID-5. A transient reconstruct operation is one that occurs on a non-degraded RAID-5 volume that has not been predicted. Limiting the number of these operations that can occur simultaneously removes the possibility of flooding the system with many reconstruct operations, and so reduces the risk of causing memory starvation.

The default number of transient reconstruct operations that can be performed in parallel is 1.

Increasing this size improves the initial performance on the system when a failure first occurs and before a detach of a failing object is performed, but can lead to memory starvation.

Appendix A

Commands summary

This appendix summarizes the usage and purpose of important commonly-used commands in Veritas Volume Manager (VxVM). References are included to longer descriptions in the remainder of this book.

Most commands (excepting daemons, library commands and supporting scripts) are linked to the `/usr/sbin` directory from the `/opt/VRTS/bin` directory. It is recommended that you add the following directories to your `PATH` environment variable:

- If you are using the Bourne or Korn shell (`sh` or `ksh`), use the commands:

```
$ PATH=$PATH:/usr/sbin:/opt/VRTS/bin:/opt/VRTSvxfs/sbin:\
/opt/VRTSdbed/bin:/opt/VRTSdb2ed/bin:/opt/VRTSsybed/bin:\
/opt/VRTSob/bin
$ MANPATH=/usr/share/man:/opt/VRTS/man:$MANPATH
$ export PATH MANPATH
```

- If you are using a C shell (`csh` or `tcsh`), use the commands:

```
% set path = ( $path /usr/sbin /opt/VRTSvxfs/sbin \
/opt/VRTSdbed/bin /opt/VRTSdb2ed/bin /opt/VRTSsybed/bin \
/opt/VRTSob/bin /opt/VRTS/bin )
% setenv MANPATH /usr/share/man:/opt/VRTS/man:$MANPATH
```

Note: If you have not installed database software, you can omit `/opt/VRTSdbed/bin`, `/opt/VRTSdb2ed/bin` and `/opt/VRTSsybed/bin`. Similarly, `/opt/VRTSvxfs/bin` is only required to access some VxFS commands.

VxVM library commands and supporting scripts are located under the `/usr/lib/vxvm` directory hierarchy. You can include these directories in your path if you need to use them on a regular basis.

For detailed information about an individual command, refer to the appropriate manual page in the 1M section. A list of manual pages is provided in "Online manual pages" on page 507. Commands and scripts that are provided to support

other commands and scripts, and which are not intended for general use, are not located in `/opt/VRTS/bin` and do not have manual pages.

The following tables summarize the commonly-used commands:

- "Obtaining information about objects in VxVM" on page 488
- "Administering disks" on page 489
- "Creating and administering disk groups" on page 492
- "Creating and administering subdisks" on page 494
- "Creating and administering plexes" on page 496
- "Creating volumes" on page 498
- "Administering volumes" on page 501
- "Monitoring and controlling tasks" on page 505

Table A-1 Obtaining information about objects in VxVM

Command	Description
`vxdctl license`	List licensed features of VxVM.
`vxdisk [-g diskgroup] list [diskname]`	Lists disks under control of VxVM. See "Displaying disk information" on page 120. Example: **# vxdisk -g mydg list**
`vxdg list [diskgroup]`	Lists information about disk groups. See "Displaying disk group information" on page 169. Example: **# vxdg list mydg**
`vxdg -s list`	Lists information about shared disk groups. See "Listing shared disk groups" on page 421. Example: **# vxdg -s list**

Table A-1 Obtaining information about objects in VxVM

Command	Description
`vxinfo [-g diskgroup] [volume ...]`	Displays information about the accessibility and usability of volumes. See "Listing Unstartable Volumes" in the *Veritas Volume Manager Troubleshooting Guide*. Example: `# vxinfo -g mydg myvol1 \ myvol2`
`vxprint -hrt [-g diskgroup] [object]`	Prints single-line information about objects in VxVM. See "Displaying volume information" on page 264. Example: `# vxprint -g mydg myvol1 \ myvol2`
`vxprint -st [-g diskgroup] [subdisk]`	Displays information about subdisks. See "Displaying subdisk information" on page 216. Example: `# vxprint -st -g mydg`
`vxprint -pt [-g diskgroup] [plex]`	Displays information about plexes. See "Displaying plex information" on page 224. Example: `# vxprint -pt -g mydg`

Table A-2 Administering disks

Command	Description
`vxdiskadm`	Administers disks in VxVM using a menu-based interface.

Table A-2 Administering disks

Command	Description
`vxdiskadd [devicename ...]`	Adds a disk specified by device name. See "Using vxdiskadd to place a disk under control of VxVM" on page 101. Example: **# vxdiskadd c0t1d0**
`vxedit [-g diskgroup] rename olddisk \ newdisk`	Renames a disk under control of VxVM. See "Renaming a disk" on page 119. Example: **# vxedit -g mydg rename \ mydg03 mydg02**
`vxedit [-g diskgroup] set \ reserve=on\|off diskname`	Sets aside/does not set aside a disk from use in a disk group. See "Reserving disks" on page 119. Examples: **# vxedit -g mydg set \ reserve=on mydg02** **# vxedit -g mydg set \ reserve=off mydg02**
`vxedit [-g diskgroup] set \ nohotuse=on\|off diskname`	Does not/does allow free space on a disk to be used for hot-relocation. See "Excluding a disk from hot-relocation use" on page 388. See "Making a disk available for hot-relocation use" on page 389. Examples: **# vxedit -g mydg set \ nohotuse=on mydg03** **# vxedit -g mydg set \ nohotuse=off mydg03**

Table A-2 Administering disks

Command	Description
`vxedit [-g diskgroup] set \ spare=on\|off diskname`	Adds/removes a disk from the pool of hot-relocation spares. See "Marking a disk as a hot-relocation spare" on page 387. See "Removing a disk from use as a hot-relocation spare" on page 388. Examples: `# vxedit -g mydg set \ spare=on mydg04` `# vxedit -g mydg set \ spare=off mydg04`
`vxdisk offline devicename`	Takes a disk offline. See "Taking a disk offline" on page 118. Example: `# vxdisk offline c0t1d0`
`vxdg -g diskgroup rmdisk diskname`	Removes a disk from its disk group. See "Removing a disk from a disk group" on page 172. Example: `# vxdg -g mydg rmdisk c0t2d0`
`vxdiskunsetup devicename`	Removes a disk from control of VxVM. See "Removing a disk from a disk group" on page 172. Example: `# vxdiskunsetup c0t3d0`

Table A-3 Creating and administering disk groups

Command	Description
`vxdg [-s] init` *diskgroup* `\` `[`*diskname*`=]` *devicename*	Creates a disk group using a pre-initialized disk. See "Creating a disk group" on page 170. See "Creating a shared disk group" on page 422. Example: **# vxdg init mydg \ mydg01=c0t1d0**
`vxsplitlines -g` *diskgroup*	Reports conflicting configuration information. See "Handling conflicting configuration copies" on page 190. Example: **# vxsplitlines -g mydg**
`vxdg [-n newname] deport` *diskgroup*	Deports a disk group and optionally renames it. See "Deporting a disk group" on page 173. Example: **# vxdg -n newdg deport mydg**
`vxdg [-n newname] import` *diskgroup*	Imports a disk group and optionally renames it. See "Importing a disk group" on page 174. Example: **# vxdg -n newdg import mydg**
`vxdg [-n newname] -s import` *diskgroup*	Imports a disk group as shared by a cluster, and optionally renames it. See "Importing disk groups as shared" on page 423. Example: **# vxdg -n newsdg -s import \ mysdg**

Table A-3 Creating and administering disk groups

Command	Description
`vxdg [-o expand] listmove` *sourcedg* \ *targetdg object ...*	Lists the objects potentially affected by moving a disk group. See "Listing objects potentially affected by a move" on page 200. Example: `# vxdg -o expand listmove \ mydg newdg myvol1`
`vxdg [-o expand] move` *sourcedg* \ *targetdg object ...*	Moves objects between disk groups. See "Moving objects between disk groups" on page 203. Example: `# vxdg -o expand move mydg \ newdg myvol1`
`vxdg [-o expand] split` *sourcedg* \ *targetdg object ...*	Splits a disk group and moves the specified objects into the target disk group. See "Splitting disk groups" on page 205. Example: `# vxdg -o expand split mydg \ newdg myvol2 myvol3`
`vxdg join` *sourcedg targetdg*	Joins two disk groups. See "Joining disk groups" on page 206. Example: `# vxdg join newdg mydg`
`vxdg -g` *diskgroup* `set \ activation=ew\|ro\|sr\|sw\|off`	Sets the activation mode of a shared disk group in a cluster. See "Changing the activation mode on a shared disk group" on page 425. Example: `# vxdg -g mysdg set \ activation=sw`

Table A-3 Creating and administering disk groups

Command	Description
`vxrecover -g` *diskgroup* `-sb`	Starts all volumes in an imported disk group. See "Moving disk groups between systems" on page 185. Example: **# vxrecover -g mydg -sb**
`vxdg destroy` *diskgroup*	Destroys a disk group and releases its disks. See "Destroying a disk group" on page 208. Example: **# vxdg destroy mydg**

Table A-4 Creating and administering subdisks

Command	Description
`vxmake [-g` *diskgroup*`] sd` *subdisk* `\` *diskname,offset,length*	Creates a subdisk. See "Creating subdisks" on page 215. Example: **# vxmake -g mydg sd \ mydg02-01 mydg02,0,8000**
`vxsd [-g` *diskgroup*`] assoc` *plex* `\` *subdisk*`...`	Associates subdisks with an existing plex. See "Associating subdisks with plexes" on page 218. Example: **# vxsd -g mydg assoc home-1 mydg02-01 \ mydg02-00 mydg02-01**

Table A-4 Creating and administering subdisks

Command	Description
`vxsd [-g` *diskgroup*`] assoc` *plex* `\` *subdisk1:0 ... subdiskM:N-1*	Adds subdisks to the ends of the columns in a striped or RAID-5 volume. See "Associating subdisks with plexes" on page 218. Example: `# vxsd -g mydg assoc \ vol01-01 mydg10-01:0 \ mydg11-01:1 mydg12-01:2`
`vxsd [-g` *diskgroup*`] mv` *oldsubdisk* `\` *newsubdisk . . .*	Replaces a subdisk. See "Moving subdisks" on page 217. Example: `# vxsd -g mydg mv mydg01-01 \ mydg02-01`
`vxsd [-g` *diskgroup*`] -s` *size* `split \` *subdisk sd1 sd2*	Splits a subdisk in two. See "Splitting subdisks" on page 217. Example: `# vxsd -g mydg -s 1000m \ split mydg03-02 mydg03-02 \ mydg03-03`
`vxsd [-g` *diskgroup*`] join` *sd1 sd2 . . .* `\` *subdisk*	Joins two or more subdisks. See "Joining subdisks" on page 218. Example: `# vxsd -g mydg join \ mydg03-02 mydg03-03 \ mydg03-02`
`vxassist [-g` *diskgroup*`] move \` *volume* `!`*olddisk* *newdisk*	Relocates subdisks in a volume between disks. See "Moving and unrelocating subdisks using vxassist" on page 392. Example: `# vxassist -g mydg move \ myvol !mydg02 mydg05`

Table A-4 Creating and administering subdisks

Command	Description
`vxunreloc [-g diskgroup] original_disk`	Relocates subdisks to their original disks. See "Moving and unrelocating subdisks using vxunreloc" on page 392. Example: `# vxunreloc -g mydg mydg01`
`vxsd [-g diskgroup] dis subdisk`	Dissociates a subdisk from a plex. See "Dissociating subdisks from plexes" on page 221. Example: `# vxsd -g mydg dis mydg02-01`
`vxedit [-g diskgroup] rm subdisk`	Removes a subdisk. See "Removing subdisks" on page 221. Example: `# vxedit -g mydg rm mydg02-01`
`vxsd [-g diskgroup] -o rm dis subdisk`	Dissociates and removes a subdisk from a plex. See "Dissociating subdisks from plexes" on page 221. Example: `# vxsd -g mydg -o rm dis \` `mydg02-01`

Table A-5 Creating and administering plexes

Command	Description
`vxmake [-g diskgroup] plex plex \` `sd=subdisk1[,subdisk2,...]`	Creates a concatenated plex. See "Creating plexes" on page 223. Example: `# vxmake -g mydg plex \` `vol01-02 \` `sd=mydg02-01,mydg02-02`

Table A-5 Creating and administering plexes

Command	Description
`vxmake [-g` *diskgroup*`] plex` *plex* `\ layout=stripe\|raid5 stwidth=`*W* `\ ncolumn=`*N* `sd=`*subdisk1*`[,`*subdisk2*`,...]`	Creates a striped or RAID-5 plex. See "Creating a striped plex" on page 224. Example: **# vxmake -g mydg plex pl-01 \ layout=stripe stwidth=32 \ ncolumn=2 \ sd=mydg01-01,mydg02-01**
`vxplex [-g` *diskgroup*`] att` *volume plex*	Attaches a plex to an existing volume. See "Attaching and associating plexes" on page 229. See "Reattaching plexes" on page 231. Example: **# vxplex -g mydg att vol01 \ vol01-02**
`vxplex [-g` *diskgroup*`] det` *plex*	Detaches a plex. See "Detaching plexes" on page 231. Example: **# vxplex -g mydg det vol01-02**
`vxmend [-g` *diskgroup*`] off` *plex*	Takes a plex offline for maintenance. See "Taking plexes offline" on page 230. Example: **# vxmend -g mydg off vol02-02**
`vxmend [-g` *diskgroup*`] on` *plex*	Re-enables a plex for use. See "Reattaching plexes" on page 231. Example: **# vxmend -g mydg on vol02-02**
`vxplex [-g` *diskgroup*`] mv` *oldplex newplex*	Replaces a plex. See "Moving plexes" on page 232. Example: **# vxplex -g mydg mv \ vol02-02 vol02-03**

Table A-5 Creating and administering plexes

Command	Description
`vxplex [-g` *diskgroup*`] cp` *volume newplex*	Copies a volume onto a plex. See "Copying volumes to plexes" on page 233. Example: `# vxplex -g mydg cp vol02 \ vol03-01`
`vxmend [-g` *diskgroup*`] fix clean` *plex*	Sets the state of a plex in an unstartable volume to CLEAN. See "Reattaching plexes" on page 231. Example: `# vxmend -g mydg fix clean \ vol02-02`
`vxplex [-g` *diskgroup*`] -o rm dis` *plex*	Dissociates and removes a plex from a volume. See "Dissociating and removing plexes" on page 233. Example: `# vxplex -g mydg -o rm dis \ vol03-01`

Table A-6 Creating volumes

Command	Description
`vxassist [-g` *diskgroup*`] maxsize \ layout=`*layout* `[`*attributes*`]`	Displays the maximum size of volume that can be created. See "Discovering the maximum size of a volume" on page 242. Example: `# vxassist -g mydg maxsize \ layout=raid5 nlog=2`

Table A-6 Creating volumes

Command	Description
`vxassist -b [-g` *diskgroup*`] make \` *volume length* `[layout=`*layout* `] [`*attributes*`]`	Creates a volume. See "Creating a volume on any disk" on page 243. See "Creating a volume on specific disks" on page 244. Example: `# vxassist -b -g mydg make \ myvol 20g layout=concat \ mydg01 mydg02`
`vxassist -b [-g` *diskgroup*`] make \` *volume length* `layout=mirror \ [nmirror=`*N*`] [`*attributes*`]`	Creates a mirrored volume. See "Creating a mirrored volume" on page 249. Example: `# vxassist -b -g mydg make \ mymvol 20g layout=mirror \ nmirror=2`
`vxassist -b [-g` *diskgroup*`] make \` *volume length* `layout=`*layout* `\ exclusive=on [`*attributes*`]`	Creates a volume that may be opened exclusively by a single node in a cluster. See "Creating volumes with exclusive open access by a node" on page 426. Example: `# vxassist -b -g mysdg make \ mysmvol 20g layout=mirror \ exclusive=on`
`vxassist -b [-g` *diskgroup*`] make \` *volume length* `layout={stripe\|raid5} \ [stripeunit=`*W*`] [ncol=`*N*`] [`*attributes*`]`	Creates a striped or RAID-5 volume. See "Creating a striped volume" on page 253. See"Creating a RAID-5 volume" on page 256. Example: `# vxassist -b -g mydg make \ mysvol 20g layout=stripe \ stripeunit=32 ncol=4`

Table A-6 Creating volumes

Command	Description
`vxassist -b [-g` *diskgroup*`] make \` *volume length* `layout=mirror \` `mirror=ctlr [`*attributes*`]`	Creates a volume with mirrored data plexes on separate controllers. See "Mirroring across targets, controllers or enclosures" on page 255. Example: `# vxassist -b -g mydg make \ mymcvol 20g layout=mirror \ mirror=ctlr`
`vxmake -b [-g` *diskgroup*`] -U`*usage_type* `\ vol` *volume* `[len=`*length*`] plex=`*plex,...*	Creates a volume from existing plexes. See "Creating a volume using vxmake" on page 258. Example: `# vxmake -g mydg -Uraid5 \ vol r5vol \ plex=raidplex,raidlog1,\ raidlog2`
`vxvol [-g` *diskgroup*`] start` *volume*	Initializes and starts a volume for use. See "Initializing and starting a volume" on page 260. See "Starting a volume" on page 271. Example: `# vxvol -g mydg start r5vol`
`vxvol [-g` *diskgroup*`] init zero` *volume*	Initializes and zeros out a volume for use. See "Initializing and starting a volume" on page 260. Example: `# vxvol -g mydg init zero \ myvol`

Table A-7 Administering volumes

Command	Description
`vxassist [-g` *diskgroup*`] mirror` *volume* `\ [`*attributes*`]`	Adds a mirror to a volume. See "Adding a mirror to a volume" on page 271. Example: `# vxassist -g mydg mirror \ myvol mydg10`
`vxassist [-g` *diskgroup*`] remove \ mirror` *volume* `[`*attributes*`]`	Removes a mirror from a volume. See "Removing a mirror" on page 273. Example: `# vxassist -g mydg remove \ mirror myvol !mydg11`
`vxassist [-g` *diskgroup*`] \ {growto\|growby}` *volume length*	Grows a volume to a specified size or by a specified amount. See "Resizing volumes using vxassist" on page 286. Example: `# vxassist -g mydg growby \ myvol 10g`
`vxassist [-g` *diskgroup*`] \ {shrinkto\|shrinkby}` *volume length*	Shrinks a volume to a specified size or by a specified amount. See "Resizing volumes using vxassist" on page 286. Example: `# vxassist -g mydg shrinkto \ myvol 20g`
`vxresize -b -F vxfs [-g` *diskgroup*`] \` *volume length diskname . . .*	Resizes a volume and the underlying Veritas File System. See "Resizing volumes using vxresize" on page 285. Example: `# vxassist -b -F vxfs \ -g mydg myvol 20g mydg10 \ mydg11`

Table A-7 Administering volumes

Command	Description
`vxsnap [-g diskgroup] prepare volume \ [drl=on\|sequential\|off]`	Prepares a volume for instant snapshots and for DRL logging. See "Preparing a volume for DRL and instant snapshots" on page 275. Example: `# vxsnap -g mydg prepare \ myvol drl=on`
`vxsnap [-g diskgroup] make \ source=volume/newvol=snapvol\ [/nmirror=number]`	Takes a full-sized instant snapshot of a volume by breaking off plexes of the original volume. See "Creating instant snapshots" on page 319. Example: `# vxsnap -g mydg make \ source=myvol/\ newvol=mysnpvol/\ nmiror=2`
`vxsnap [-g diskgroup] make \ source=volume/snapvol=snapvol`	Takes a full-sized instant snapshot of a volume using a prepared empty volume. See "Creating a volume for use as a full-sized instant or linked break-off snapshot" on page 323. See "Creating instant snapshots" on page 319. Example: `# vxsnap -g mydg make \ source=myvol/snapvol=snpvol`

Table A-7 Administering volumes

Command	Description
`vxmake [-g diskgroup] cache \ cache_object cachevolname=volume \ [regionsize=size]`	Creates a cache object for use by space-optimized instant snapshots. See "Creating a shared cache object" on page 322. A cache volume must have already been created, as shown in this example: `# vxassist -g mydg make \ cvol 1g layout=mirror \ init=active mydg16 mydg17` `# vxmake -g mydg cache cobj \ cachevolname=cvol`
`vxsnap [-g diskgroup] make \ source=volume/newvol=snapvol\ /cache=cache_object`	Takes a space-optimized instant snapshot of a volume. See "Creating instant snapshots" on page 319. Example: `# vxsnap -g mydg make \ source=myvol/\ newvol=mysosvol/\ cache=cobj`
`vxsnap [-g diskgroup] refresh snapshot`	Refreshes a snapshot from its original volume. See "Refreshing an instant snapshot" on page 337. Example: `# vxsnap -g mydg refresh \ mysnpvol`
`vxsnap [-g diskgroup] dis snapshot`	Turns a snapshot into an independent volume. See "Dissociating an instant snapshot" on page 340. Example: `# vxsnap -g mydg dis mysnpvol`

Table A-7 Administering volumes

Command	Description
`vxsnap [-g diskgroup] unprepare volume`	Removes support for instant snapshots and DRL logging from a volume. See "Removing support for DRL and instant snapshots from a volume" on page 279. Example: `# vxsnap -g mydg unprepare \ myvol`
`vxassist [-g diskgroup] relayout \ volume [layout=layout] [relayout_options]`	Performs online relayout of a volume. See "Performing online relayout" on page 294. Example: `# vxassist -g mydg relayout \ vol2 layout=stripe`
`vxassist [-g diskgroup] relayout \ volume layout=raid5 stripeunit=W \ ncol=N`	Relays out a volume as a RAID-5 volume with stripe width *W* and *N* columns. See "Performing online relayout" on page 294. Example: `# vxassist -g mydg relayout \ vol3 layout=raid5 \ stripeunit=16 ncol=4`
`vxrelayout [-g diskgroup] -o bg \ reverse volume`	Reverses the direction of a paused volume relayout. See "Controlling the progress of a relayout" on page 299. Example: `# vxrelayout -g mydg -o bg \ reverse vol3`

Table A-7 Administering volumes

Command	Description
`vxassist [-g` *diskgroup*`] convert \` *volume* `[layout=`*layout*`] [`*convert_options*`]`	Converts between a layered volume and a non-layered volume layout. See "Converting between layered and non-layered volumes" on page 300. Example: `# vxassist -g mydg convert \ vol3 layout=stripe-mirror`
`vxassist [-g` *diskgroup*`] remove \ volume` *volume*	Removes a volume. See "Removing a volume" on page 290. Example: `# vxassist -g mydg remove \ myvol`

Table A-8 Monitoring and controlling tasks

Command	Description
`command [-g` *diskgroup*`] -t tasktag \ [options] [arguments]`	Specifies a task tag to a VxVM command. See "Specifying task tags" on page 267. Example: `# vxrecover -g mydg \ -t mytask -b mydg05`
`vxtask [-h] [-g` *diskgroup*`] list`	Lists tasks running on a system. See "Using the vxtask command" on page 269. Example: `# vxtask -h -g mydg list`
`vxtask monitor` *task*	Monitors the progress of a task. See "Using the vxtask command" on page 269. Example: `# vxtask monitor mytask`

Table A-8 Monitoring and controlling tasks

Command	Description
`vxtask pause` *task*	Suspends operation of a task. See "Using the vxtask command" on page 269. Example: **# vxtask pause mytask**
`vxtask -p [-g` *diskgroup*`] list`	Lists all paused tasks. See "Using the vxtask command" on page 269. Example: **# vxtask -p -g mydg list**
`vxtask resume` *task*	Resumes a paused task. See "Using the vxtask command" on page 269. Example: **# vxtask resume mytask**
`vxtask abort` *task*	Cancels a task and attempts to reverse its effects. See "Using the vxtask command" on page 269. Example: **# vxtask abort mytask**

Online manual pages

Manual pages are organized into three sections:

- Section 1M – administrative commands
- Section 4 – file formats
- Section 7 – device driver interfaces

Section 1M — administrative commands

Manual pages in section 1M describe commands that are used to administer Veritas Volume Manager.

Table A-9 Section 1M manual pages

Name	Description
dgcfgbackup	Create or update VxVM volume group configuration backup file.
dgcfgdaemon	Start the VxVM configuration backup daemon.
dgcfgrestore	Display or restore VxVM disk group configuration from backup.
vgrestore	Restore a VxVM disk group back to an LVM volume group.
vx_emerg_start	Start Veritas Volume Manager from recovery media.
vxassist	Create, relayout, convert, mirror, backup, grow, shrink, delete, and move volumes.
vxbootsetup	Set up system boot information on a Veritas Volume Manager disk.
vxbrk_rootmir	Break off a mirror of a VxVM root disk to create a separate root disk generation.
vxcache	Administer the cache object for space-optimized snapshots.
vxcached	Resize cache volumes when required.
vxcdsconvert	Make disks and disk groups portable between systems.
vxchg_rootid	Set up VxVM root disk that has been cloned or copied from another operational VxVM root disk.
vxclustadm	Start, stop, and reconfigure a cluster.
vxcmdlog	Administer command logging.
vxconfigbackup	Back up disk group configuration.
vxconfigbackupd	Disk group configuration backup daemon.

Table A-9 Section 1M manual pages

Name	Description
vxconfigd	Veritas Volume Manager configuration daemon
vxconfigrestore	Restore disk group configuration.
vxcp_lvmroot	Copy LVM root disk onto new Veritas Volume Manager root disk.
vxdarestore	Restore simple or nopriv disk access records.
vxdco	Perform operations on version 0 DCO objects and DCO volumes.
vxdctl	Control the volume configuration daemon.
vxddladm	Device Discovery Layer subsystem administration.
vxdestroy_lvmroot	Remove LVM root disk and associated LVM volume group.
vxdg	Manage Veritas Volume Manager disk groups.
vxdisk	Define and manage Veritas Volume Manager disks.
vxdiskadd	Add one or more disks for use with Veritas Volume Manager.
vxdiskadm	Menu-driven Veritas Volume Manager disk administration.
vxdisksetup	Configure a disk for use with Veritas Volume Manager.
vxdiskunsetup	Deconfigure a disk from use with Veritas Volume Manager.
vxdmpadm	DMP subsystem administration.
vxdmpinq	Display SCSI inquiry data.
vxedit	Create, remove, and modify Veritas Volume Manager records.
vxevac	Evacuate all volumes from a disk.
vximportdg	Import a disk group into the Veritas Volume Manager configuration.
vxinfo	Print accessibility and usability of volumes.
vxinstall	Menu-driven Veritas Volume Manager initial configuration.
vxintro	Introduction to the Veritas Volume Manager utilities.
vxiod	Start, stop, and report on Veritas Volume Manager kernel I/O threads.
vxmake	Create Veritas Volume Manager configuration records.
vxmemstat	Display memory statistics for Veritas Volume Manager.

Table A-9 Section 1M manual pages

Name	Description
vxmend	Mend simple problems in configuration records.
vxmirror	Mirror volumes on a disk or control default mirroring.
vxnotify	Display Veritas Volume Manager configuration events.
vxpfto	Set Powerfail Timeout (pfto).
vxplex	Perform Veritas Volume Manager operations on plexes.
vxpool	Create and administer ISP storage pools.
vxprint	Display records from the Veritas Volume Manager configuration.
vxr5check	Verify RAID-5 volume parity.
vxreattach	Reattach disk drives that have become accessible again.
vxrecover	Perform volume recovery operations.
vxrelayout	Convert online storage from one layout to another.
vxrelocd	Monitor Veritas Volume Manager for failure events and relocate failed subdisks.
vxres_lvmroot	Restore LVM root disk from Veritas Volume Manager root disk.
vxresize	Change the length of a volume containing a file system.
vxrootmir	Create a mirror of a Veritas Volume Manager root disk.
vxsd	Perform Veritas Volume Manager operations on subdisks.
vxse	Storage Expert rules.
vxsnap	Enable DRL on a volume, and create and administer instant snapshots.
vxsplitlines	Show disks with conflicting configuration copies in a cluster.
vxstat	Veritas Volume Manager statistics management utility.
vxtask	List and administer Veritas Volume Manager tasks.
vxtemplate	Install and administer ISP volume templates and template sets.
vxtrace	Trace operations on volumes.
vxtranslog	Administer transaction logging.

Table A-9 Section 1M manual pages

Name	Description
vxtune	Adjust Veritas Volume Replicator and Veritas Volume Manager tunables.
vxunreloc	Move a hot-relocated subdisk back to its original disk.
vxusertemplate	Create and administer ISP user templates.
vxvmboot	Prepare Veritas Volume Manager volume as a root, boot, primary swap or dump volume.
vxvmconvert	Convert LVM volume groups to VxVM disk groups.
vxvol	Perform Veritas Volume Manager operations on volumes.
vxvoladm	Create and administer ISP application volumes on allocated storage.
vxvoladmtask	Administer ISP tasks.
vxvset	Create and administer volume sets.

Section 4 — file formats

Manual pages in section 4 describe the format of files that are used by Veritas Volume Manager.

Table A-10 Section 4 manual pages

Name	Description
vol_pattern	Disk group search specifications.
vxmake	vxmake description file.

Section 7 — device driver interfaces

Manual pages in section 7 describe the interfaces to Veritas Volume Manager devices.

Appendix B

Configuring Veritas Volume Manager

This appendix provides guidelines for setting up efficient storage management after installing the Veritas Volume Manager software.

This chapter describes:

- Setup tasks after installation
- Adding unsupported disk arrays as JBODs
- Adding foreign devices
- Adding disks to disk groups
- Guidelines for configuring storage
- Controlling VxVM's view of multipathed devices
- Configuring cluster support
- Reconfiguration tasks

Setup tasks after installation

The setup sequence listed below is a typical example. Your system requirements may differ.

Initial Setup Tasks

- Create disk groups by placing disks under Veritas Volume Manager control.
- If you intend to use the Intelligent Storage Provisioning (ISP) feature, create storage pools within the disk groups.
- Create volumes in the disk groups.
- Configure file systems on the volumes.

Optional Setup Tasks

- Place the `root` disk under VxVM control and mirror it to create an alternate boot disk.
- Designate hot-relocation spare disks in each disk group.
- Add mirrors to volumes.
- Configure DRL and FastResync on volumes.

Maintenance Tasks

- Resize volumes and file systems.
- Add more disks, create new disk groups, and create new volumes.
- Create and maintain snapshots.

Adding unsupported disk arrays as JBODs

After installation, add any disk arrays that are unsupported by Symantec to the DISKS (JBOD) category as described in "Administering the Device Discovery Layer" on page 85.

Adding foreign devices

The device discovery feature of VxVM can discover some devices that are controlled by third-party drivers, such as for EMC PowerPath. For these devices it may be preferable to use the multipathing capability that is provided by the third-party drivers rather than using the Dynamic Multipathing (DMP) feature. Provided that a suitable array support library is available, DMP can co-exist with such drivers. Other foreign devices, for which a compatible ASL does not exist, can be made available to Veritas Volume Manager as simple disks by using the `vxddladm addforeign` command. This also has the effect of bypassing DMP. Refer to "Administering the Device Discovery Layer" on page 85 for more information.

Adding disks to disk groups

To place disks in disk groups, use VEA or the `vxdiskadm` program after completing the installation. Refer to "Adding a disk to VxVM" on page 97 and the VEA online help for information on how to add your disks to new disk groups.

See the *Veritas Storage Foundation Intelligent Storage Provisioning Administrator's Guide* for information about creating storage pools within disk

groups. Storage pools are only required if you intend using the ISP feature of VxVM.

Guidelines for configuring storage

A disk failure can cause loss of data on the failed disk and loss of access to your system. Loss of access is due to the failure of a key disk used for system operations. Veritas Volume Manager can protect your system from these problems.

To maintain system availability, data important to running and booting your system must be mirrored. The data must be preserved so it can be used in case of failure.

The following are suggestions for protecting your system and data:

- Perform regular backups to protect your data. Backups are necessary if all copies of a volume are lost or corrupted. Power surges can damage several (or all) disks on your system. Also, typing a command in error can remove critical files or damage a file system directly. Performing regular backups ensures that lost or corrupted data is available to be retrieved.
- Place the disk containing the `root` file system (the root or boot disk) under Veritas Volume Manager control. Mirror the root disk so that an alternate root disk exists for booting purposes. By mirroring disks critical to booting, you ensure that no single disk failure leaves your system unbootable and unusable. For more information, see “Rootability” on page 102.
- Use mirroring to protect data against loss from a disk failure. See “Mirroring guidelines” on page 514 for details.
- Use the DRL feature to speed up recovery of mirrored volumes after a system crash. See “Dirty region logging guidelines” on page 515 for details.
- Use striping to improve the I/O performance of volumes. See “Striping guidelines” on page 515 for details.
- Make sure enough disks are available for a combined striped and mirrored configuration. At least two disks are required for the striped plex, and one or more additional disks are needed for the mirror.
- When combining striping and mirroring, never place subdisks from one plex on the same physical disk as subdisks from the other plex.
- Use logging to prevent corruption of recovery data in RAID-5 volumes. Make sure that each RAID-5 volume has at least one log plex. See “RAID-5 guidelines” on page 516 for details.

- Leave the Veritas Volume Manager hot-relocation feature enabled. See "Hot-relocation guidelines" on page 516 for details.

Mirroring guidelines

Refer to the following guidelines when using mirroring.

- Do not place subdisks from different plexes of a mirrored volume on the same physical disk. This action compromises the availability benefits of mirroring and degrades performance. Using the `vxassist` or `vxdiskadm` commands precludes this from happening.
- To provide optimum performance improvements through the use of mirroring, at least 70 percent of physical I/O operations should be read operations. A higher percentage of read operations results in even better performance. Mirroring may not provide a performance increase or may even result in a performance decrease in a write-intensive workload environment.

Note: The operating system implements a file system cache. Read requests can frequently be satisfied from the cache. This can cause the read/write ratio for physical I/O operations through the file system to be biased toward writing (when compared to the read/write ratio at the application level).

- Where possible, use disks attached to different controllers when mirroring or striping. Most disk controllers support overlapped seeks. This allows seeks to begin on two disks at once. Do not configure two plexes of the same volume on disks that are attached to a controller that does not support overlapped seeks. This is important for older controllers or SCSI disks that do not cache on the drive. It is less important for modern SCSI disks and controllers. Mirroring across controllers allows the system to survive a failure of one of the controllers. Another controller can continue to provide data from a mirror.
- A plex exhibits superior performance when striped or concatenated across multiple disks, or when located on a much faster device. Set the read policy to prefer the faster plex. By default, a volume with one striped plex is configured to prefer reading from the striped plex.

For more information, see "Mirroring (RAID-1)" on page 42.

Dirty region logging guidelines

Dirty region logging (DRL) can speed up recovery of mirrored volumes following a system crash. When DRL is enabled, Veritas Volume Manager keeps track of the regions within a volume that have changed as a result of writes to a plex.

Note: Using Dirty Region Logging can impact system performance in a write-intensive environment.

For more information, see "Dirty region logging" on page 60.

Striping guidelines

Refer to the following guidelines when using striping.

- Do not place more than one column of a striped plex on the same physical disk.
- Calculate stripe-unit sizes carefully. In general, a moderate stripe-unit size (for example, 64 kilobytes, which is also the default used by `vxassist`) is recommended.
- If it is not feasible to set the stripe-unit size to the track size, and you do not know the application I/O pattern, use the default stripe-unit size.

Note: Many modern disk drives have *variable geometry*. This means that the track size differs between cylinders, so that outer disk tracks have more sectors than inner tracks. It is therefore not always appropriate to use the track size as the stripe-unit size. For these drives, use a moderate stripe-unit size (such as 64 kilobytes), unless you know the I/O pattern of the application.

- Volumes with small stripe-unit sizes can exhibit poor sequential I/O latency if the disks do not have synchronized spindles. Generally, striping over disks without synchronized spindles yields better performance when used with larger stripe-unit sizes and multi-threaded, or largely asynchronous, random I/O streams.
- Typically, the greater the number of physical disks in the stripe, the greater the improvement in I/O performance; however, this reduces the effective mean time between failures of the volume. If this is an issue, combine striping with mirroring to combine high-performance with improved reliability.
- If only one plex of a mirrored volume is striped, set the policy of the volume to `prefer` for the striped plex. (The default read policy, `select`, does this automatically.)

- If more than one plex of a mirrored volume is striped, configure the same stripe-unit size for each striped plex.
- Where possible, distribute the subdisks of a striped volume across drives connected to different controllers and buses.
- Avoid the use of controllers that do not support overlapped seeks. (Such controllers are rare.)

The `vxassist` command automatically applies and enforces many of these rules when it allocates space for striped plexes in a volume.

For more information, see "Striping (RAID-0)" on page 38.

RAID-5 guidelines

Refer to the following guidelines when using RAID-5.

In general, the guidelines for mirroring and striping together also apply to RAID-5. The following guidelines should also be observed with RAID-5:

- Only one RAID-5 plex can exist per RAID-5 volume (but there can be multiple log plexes).
- The RAID-5 plex must be derived from at least three subdisks on three or more physical disks. If any log plexes exist, they must belong to disks other than those used for the RAID-5 plex.
- RAID-5 logs can be mirrored and striped.
- If the volume length is not explicitly specified, it is set to the length of any RAID-5 plex associated with the volume; otherwise, it is set to zero. If you specify the volume length, it must be a multiple of the stripe-unit size of the associated RAID-5 plex, if any.
- If the log length is not explicitly specified, it is set to the length of the smallest RAID-5 log plex that is associated, if any. If no RAID-5 log plexes are associated, it is set to zero.
- Sparse RAID-5 log plexes are not valid.
- RAID-5 volumes are not supported for sharing in a cluster.

For more information, see "RAID-5 (striping with parity)" on page 45.

Hot-relocation guidelines

Hot-relocation automatically restores redundancy and access to mirrored and RAID-5 volumes when a disk fails. This is done by relocating the affected subdisks to disks designated as spares and/or free space in the same disk group.

The hot-relocation feature is enabled by default. The associated daemon, `vxrelocd`, is automatically started during system startup.

Refer to the following guidelines when using hot-relocation.

- The hot-relocation feature is enabled by default. Although it is possible to disable hot-relocation, it is advisable to leave it enabled. It will notify you of the nature of the failure, attempt to relocate any affected subdisks that are redundant, and initiate recovery procedures.
- Although hot-relocation does not require you to designate disks as spares, designate at least one disk as a spare within each disk group. This gives you some control over which disks are used for relocation. If no spares exist, Veritas Volume Manager uses any available free space within the disk group. When free space is used for relocation purposes, it is possible to have performance degradation after the relocation.
- After hot-relocation occurs, designate one or more additional disks as spares to augment the spare space. Some of the original spare space may be occupied by relocated subdisks.
- If a given disk group spans multiple controllers and has more than one spare disk, set up the spare disks on different controllers (in case one of the controllers fails).
- For a mirrored volume, configure the disk group so that there is at least one disk that does not already contain a mirror of the volume. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- For a mirrored and striped volume, configure the disk group so that at least one disk does not already contain one of the mirrors of the volume or another subdisk in the striped plex. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- For a RAID-5 volume, configure the disk group so that at least one disk does not already contain the RAID-5 plex (or one of its log plexes) of the volume. This disk should either be a spare disk with some available space or a regular disk with some free space and the disk is not excluded from hot-relocation use.
- If a mirrored volume has a DRL log subdisk as part of its data plex, you cannot relocate the data plex. Instead, place log subdisks in log plexes that contain no data.
- Hot-relocation does not guarantee to preserve the original performance characteristics or data layout. Examine the locations of newly-relocated

subdisks to determine whether they should be relocated to more suitable disks to regain the original performance benefits.

- Although it is possible to build Veritas Volume Manager objects on spare disks (using `vxmake` or the VEA interface), it is recommended that you use spare disks for hot-relocation only.

See "Administering hot-relocation" on page 379 for more information.

Accessing volume devices

As soon as a volume has been created and initialized, it is available for use as a virtual disk partition by the operating system for the creation of a file system, or by application programs such as relational databases and other data management software.

Creating a volume in a disk group sets up block and character (raw) device files that can be used to access the volume:

`/dev/vx/dsk/`*diskgroup*`/`*volume*	block device file for *volume*
`/dev/vx/rdsk/`*diskgroup*`/`*volume*	character device file for *volume*

The pathnames include a directory named for the disk group. Use the appropriate device node to create, mount and repair file systems, and to lay out databases that require raw partitions.

Controlling VxVM's view of multipathed devices

To control how a device is treated by the Dynamic Multipathing (DMP) feature of VxVM, use the `vxdiskadm` command as described in "Disabling and enabling multipathing for specific devices" on page 133.

Configuring cluster support

The Veritas Volume Manager software includes an *optional* cluster feature that enables it to be used in a cluster environment. The cluster functionality in Veritas Volume Manager allows multiple hosts to simultaneously access and manage a set of disks under Veritas Volume Manager control. A *cluster* is a set of hosts sharing a set of disks; each host is referred to as a *node* in the cluster.

Note: The Veritas Volume Manager cluster feature requires a license, which can be obtained from your Customer Support channel.

For information about enabling cluster functionality in Veritas Volume Manager, refer to the *Veritas Storage Solutions Getting Started Guide*.

Configuring shared disk groups

This section describes how to configure shared disks in a cluster. If you are installing Veritas Volume Manager for the first time or adding disks to an existing cluster, you need to configure new shared disks.

If you are setting up Veritas Volume Manager for the first time, configure the shared disks using the following procedure:

1. Start the cluster on one node only to prevent access by other nodes.
2. On one node, run the `vxdiskadm` program and choose option 1 to initialize new disks. When asked to add these disks to a disk group, choose `none` to leave the disks for future use.
3. On other nodes in the cluster, run `vxdctl enable` to see the newly initialized disks.
4. From the master node, create disk groups on the shared disks. To determine if a node is a master or slave, run the command `vxdctl -c mode`.
 Use the `vxdg` command or VEA to create disk groups. If you use the `vxdg` command, specify the `-s` option to create shared disk groups.
5. From the master node only, use `vxassist` or VEA to create volumes in the disk groups.

 Note: RAID-5 volumes are not supported for sharing in a cluster.

6. If the cluster is only running with one node, bring up the other cluster nodes. Enter the `vxdg list` command on each node to display the shared disk groups.

Converting existing VxVM disk groups to shared disk groups

To convert existing disk groups to shared disk groups:

1. Start the cluster on one node only to prevent access by other nodes.
2. Configure the disk groups using the following procedure.
 To list all disk groups, use the following command:

   ```
   # vxdg list
   ```

 To deport the disk groups that are to be shared, use the following command:

   ```
   # vxdg deport diskgroup
   ```

 To import disk groups to be shared, use the following command:

   ```
   # vxdg -s import diskgroup
   ```

 This procedure marks the disks in the shared disk groups as shared and stamps them with the ID of the cluster, enabling other nodes to recognize the shared disks.

If dirty region logs exist, ensure they are active. If not, replace them with larger ones.

To display the shared flag for all the shared disk groups, use the following command:

```
# vxdg list
```

The disk groups are now ready to be shared.

3 Bring up the other cluster nodes. Enter the `vxdg list` command on each node to display the shared disk groups. This command displays the same list of shared disk groups displayed earlier.

For information on converting disk groups in a Veritas Storage Foundation Cluster File System environment, see the *Veritas Storage Foundation Cluster File System Installation Guide*.

Reconfiguration tasks

The following sections describe tasks that allow you to make changes to the configuration that you specified during installation.

Changing the name of the default disk group

If you use the Veritas installer to install the Veritas Volume Manager software, you can enter the name of the default disk group. This disk group will be used by commands if you do not specify the -g option, or the `VXVM_DEFAULTDG` environment variable is not set. If required, you can use the `vxdctl defaultdg` command to change the default disk group. See "Displaying and specifying the system-wide default disk group" on page 168 for details.

Enabling or disabling enclosure-based naming

If you use the Veritas installer to install the Veritas Volume Manager software, you can choose whether the displayed disk access names are based on device names or on names that you assign to disk enclosures. If required, you can use the `vxdiskadm` or `vxddladm` commands to change the naming convention that is used. See "Changing the disk-naming scheme" on page 91 for more information.

Glossary

Active/Active disk arrays

This type of multipathed disk array allows you to access a disk in the disk array through all the paths to the disk simultaneously, without any performance degradation.

Active/Passive disk arrays

This type of multipathed disk array allows one path to a disk to be designated as primary and used to access the disk at any time. Using a path other than the designated active path results in severe performance degradation in some disk arrays. Also see path, primary path, and secondary path.

associate

The process of establishing a relationship between VxVM objects; for example, a subdisk that has been created and defined as having a starting point within a plex is referred to as being associated with that plex.

associated plex

A plex associated with a volume.

associated subdisk

A subdisk associated with a plex.

atomic operation

An operation that either succeeds completely or fails and leaves everything as it was before the operation was started. If the operation succeeds, all aspects of the operation take effect at once and the intermediate states of change are invisible. If any aspect of the operation fails, then the operation aborts without leaving partial changes.

In a cluster, an atomic operation takes place either on all nodes or not at all.

attached

A state in which a VxVM object is both associated with another object and enabled for use.

block

The minimum unit of data transfer to or from a disk or array.

boot disk

A disk that is used for the purpose of booting a system.

boot disk group

A private disk group that contains the disks from which the system may be booted.

bootdg

A reserved disk group name that is an alias for the name of the boot disk group.

clean node shutdown

The ability of a node to leave a cluster gracefully when all access to shared volumes has ceased.

cluster
A set of hosts (each termed a node) that share a set of disks.

cluster manager
An externally-provided daemon that runs on each node in a cluster. The cluster managers on each node communicate with each other and inform VxVM of changes in cluster membership.

cluster-shareable disk group
A disk group in which access to the disks is shared by multiple hosts (also referred to as a shared disk group). Also see private disk group.

column
A set of one or more subdisks within a striped plex. Striping is achieved by allocating data alternately and evenly across the columns within a plex.

concatenation
A layout style characterized by subdisks that are arranged sequentially and contiguously.

configuration copy
A single copy of a configuration database.

configuration database
A set of records containing detailed information on existing VxVM objects (such as disk and volume attributes).

data change object (DCO)
A VxVM object that is used to manage information about the FastResync maps in the DCO volume. Both a DCO object and a DCO volume must be associated with a volume to implement Persistent FastResync on that volume.

data stripe
This represents the usable data portion of a stripe and is equal to the stripe minus the parity region.

DCO volume
A special volume that is used to hold Persistent FastResync change maps, and dirty region logs (see dirty region logging).

detached
A state in which a VxVM object is associated with another object, but not enabled for use.

device name
The device name or address used to access a physical disk, such as `c0t0d0`. The `c#t#d#` syntax identifies the controller, target address, and disk. In a SAN environment, it is more convenient to use *enclosure-based naming*, which forms the device name by concatenating the name of the enclosure (such as `enc0`) with the disk's number within the enclosure, separated by an underscore (for example, `enc0_2`). The term disk access name can also be used to refer to a device name.

dirty region logging
The method by which the VxVM monitors and logs modifications to a plex as a bitmap of changed regions. For a volumes with a new-style DCO volume, the dirty region log (DRL) is

maintained in the DCO volume. Otherwise, the DRL is allocated to an associated subdisk called a *log subdisk*.

disabled path

A path to a disk that is not available for I/O. A path can be *disabled* due to real hardware failures or if the user has used the `vxdmpadm disable` command on that controller.

disk

A collection of read/write data blocks that are indexed and can be accessed fairly quickly. Each disk has a universally unique identifier.

disk access name

An alternative term for a device name.

disk access records

Configuration records used to specify the access path to particular disks. Each disk access record contains a name, a type, and possibly some type-specific information, which is used by VxVM in deciding how to access and manipulate the disk that is defined by the disk access record.

disk array

A collection of disks logically arranged into an object. Arrays tend to provide benefits such as redundancy or improved performance. Also see disk enclosure and JBOD.

disk array serial number

This is the serial number of the disk array. It is usually printed on the disk array cabinet or can be obtained by issuing a vendor- specific SCSI command to the disks on the disk array. This number is used by the DMP subsystem to uniquely identify a disk array.

disk controller

In the multipathing subsystem of VxVM, the controller (host bus adapter or HBA) or disk array connected to the host, which the operating system represents as the parent node of a disk.

disk enclosure

An intelligent disk array that usually has a backplane with a built-in Fibre Channel loop, and which permits hot-swapping of disks.

disk group

A collection of disks that share a common configuration. A disk group configuration is a set of records containing detailed information on existing VxVM objects (such as disk and volume attributes) and their relationships. Each disk group has an administrator-assigned name and an internally defined unique ID. The disk group names `bootdg` (an alias for the boot disk group), `defaultdg` (an alias for the default disk group) and `nodg` (represents no disk group) are reserved.

disk group ID

A unique identifier used to identify a disk group.

disk ID

A universally unique identifier that is given to each disk and can be used to identify the disk, even if it is moved.

disk media name

An alternative term for a disk name.

disk media record

A configuration record that identifies a particular disk, by disk ID, and gives that disk a logical (or administrative) name.

disk name

A logical or administrative name chosen for a disk that is under the control of VxVM, such as `disk03`. The term disk media name is also used to refer to a disk name.

dissociate

The process by which any link that exists between two VxVM objects is removed. For example, dissociating a subdisk from a plex removes the subdisk from the plex and adds the subdisk to the free space pool.

dissociated plex

A plex dissociated from a volume.

dissociated subdisk

A subdisk dissociated from a plex.

distributed lock manager

A lock manager that runs on different systems in a cluster, and ensures consistent access to distributed resources.

enabled path

A path to a disk that is available for I/O.

encapsulation

A process that converts existing partitions on a specified disk to volumes. Encapsulation does not apply to HP-UX.

enclosure

See disk enclosure.

enclosure-based naming

See device name.

fabric mode disk

A disk device that is accessible on a Storage Area Network (SAN) via a Fibre Channel switch.

FastResync

A fast resynchronization feature that is used to perform quick and efficient resynchronization of stale mirrors, and to increase the efficiency of the snapshot mechanism. Also see Persistent FastResync and Non-Persistent FastResync.

Fibre Channel

A collective name for the fiber optic technology that is commonly used to set up a Storage Area Network (SAN).

file system

A collection of files organized together into a structure. The UNIX file system is a hierarchical structure consisting of directories and files.

free space

An area of a disk under VxVM control that is not allocated to any subdisk or reserved for use by any other VxVM object.

free subdisk

A subdisk that is not associated with any plex and has an empty `putil[0]` field.

hostid

A string that identifies a host to VxVM. The *hostid* for a host is stored in its volboot file, and is used in defining ownership of disks and disk groups.

hot-relocation

A technique of automatically restoring redundancy and access to mirrored and RAID-5 volumes when a disk fails. This is done by relocating the affected subdisks to disks designated as spares and/or free space in the same disk group.

hot-swap

Refers to devices that can be removed from, or inserted into, a system without first turning off the power supply to the system.

initiating node

The node on which the system administrator is running a utility that requests a change to VxVM objects. This node initiates a volume reconfiguration.

JBOD

The common name for an unintelligent disk array which may, or may not, support the hot-swapping of disks. The name is derived from "just a bunch of disks."

log plex

A plex used to store a RAID-5 log. The term *log plex* may also be used to refer to a Dirty Region Logging plex.

log subdisk

A subdisk that is used to store a dirty region log.

master node

A node that is designated by the software to coordinate certain VxVM operations in a cluster. Any node is capable of being the master node.

mastering node

The node to which a disk is attached. This is also known as a *disk owner*.

mirror

A duplicate copy of a volume and the data therein (in the form of an ordered collection of subdisks). Each mirror consists of one *plex* of the volume with which the mirror is associated.

mirroring

A layout technique that mirrors the contents of a volume onto multiple plexes. Each plex duplicates the data stored on the volume, but the plexes themselves may have different layouts.

multipathing

Where there are multiple physical access paths to a disk connected to a system, the disk is called multipathed. Any software residing on the host, (for example, the DMP driver) that hides this fact from the user is said to provide multipathing functionality.

node

One of the hosts in a cluster.

node abort

A situation where a node leaves a cluster (on an emergency basis) without attempting to stop ongoing operations.

node join

The process through which a node joins a cluster and gains access to shared disks.

Non-Persistent FastResync

A form of FastResync that cannot preserve its maps across reboots of the system because it stores its change map in memory.

object

An entity that is defined to and recognized internally by VxVM. The VxVM objects are: volume, plex, subdisk, disk, and disk group. There are actually two types of disk objects—one for the physical aspect of the disk and the other for the logical aspect.

parity

A calculated value that can be used to reconstruct data after a failure. While data is being written to a RAID-5 volume, parity is also calculated by performing an *exclusive OR* (XOR) procedure on data. The resulting parity is then written to the volume. If a portion of a RAID-5 volume fails, the data that was on that portion of the failed volume can be recreated from the remaining data and the parity.

parity stripe unit

A RAID-5 volume storage region that contains parity information. The data contained in the parity stripe unit can be used to help reconstruct regions of a RAID-5 volume that are missing because of I/O or disk failures.

partition

The standard division of a physical disk device, as supported directly by the operating system and disk drives.

path

When a disk is connected to a host, the path to the disk consists of the HBA (Host Bus Adapter) on the host, the SCSI or fibre cable connector and the controller on the disk or disk array. These components constitute a path to a disk. A failure on any of these results in DMP trying to shift all I/O for that disk onto the remaining (alternate) paths. Also see Active/Passive disk arrays, primary path and secondary path.

pathgroup

In the case of disks which are not multipathed by `vxdmp`, VxVM will see each path as a disk. In such cases, all paths to the disk can be grouped. This way only one of the paths from the group is made visible to VxVM.

Persistent FastResync

A form of FastResync that can preserve its maps across reboots of the system by storing its change map in a DCO volume on disk. Also see data change object (DCO).

persistent state logging

A logging type that ensures that only active mirrors are used for recovery purposes and prevents failed mirrors from being selected for recovery. This is also known as *kernel logging*.

physical disk

The underlying storage device, which may or may not be under VxVM control.

plex

A plex is a logical grouping of subdisks that creates an area of disk space independent of physical disk size or other restrictions. Mirroring is set up by creating multiple data plexes for a single volume. Each data plex in a mirrored volume contains an identical copy of the volume data. Plexes may also be created to represent concatenated, striped and RAID-5 volume layouts, and to store volume logs.

primary path

In Active/Passive disk arrays, a disk can be bound to one particular controller on the disk array or owned by a controller. The disk can then be accessed using the path through this particular controller. Also see path and secondary path.

private disk group

A disk group in which the disks are accessed by only one specific host in a cluster. Also see shared disk group.

private region

A region of a physical disk used to store private, structured VxVM information. The *private region* contains a disk header, a table of contents, and a configuration database. The table of contents maps the contents of the disk. The disk header contains a disk ID. All data in the private region is duplicated for extra reliability.

public region

A region of a physical disk managed by VxVM that contains available space and is used for allocating subdisks.

RAID

A Redundant Array of Independent Disks (RAID) is a disk array set up with part of the combined storage capacity used for storing duplicate information about the data stored in that array. This makes it possible to regenerate the data if a disk failure occurs.

read-writeback mode

A recovery mode in which each read operation recovers plex consistency for the region covered by the read. Plex consistency is recovered by reading data from blocks of one plex and writing the data to all other writable plexes.

root configuration

The configuration database for the root disk group. This is special in that it always contains records for other disk groups, which are used for backup purposes only. It also contains disk records that define all disk devices on the system.

root disk

The disk containing the root file system. This disk may be under VxVM control.

root file system

The initial file system mounted as part of the UNIX kernel startup sequence.

root partition

The disk region on which the root file system resides.

root volume

The VxVM volume that contains the root file system, if such a volume is designated by the system configuration.

rootability

The ability to place the `root` file system and the `swap` device under VxVM control. The resulting volumes can then be mirrored to provide redundancy and allow recovery in the event of disk failure.

secondary path

In Active/Passive disk arrays, the paths to a disk other than the primary path are called secondary paths. A disk is supposed to be accessed only through the primary path until it fails, after which ownership of the disk is transferred to one of the secondary paths. Also see path and primary path.

sector

A unit of size, which can vary between systems. Sector size is set per device (hard drive, CD-ROM, and so on). Although all devices within a system are usually configured to the same sector size for interoperability, this is not always the case. A sector is commonly 1024 bytes.

shared disk group

A disk group in which access to the disks is shared by multiple hosts (also referred to as a cluster-shareable disk group). Also see private disk group.

shared volume

A volume that belongs to a shared disk group and is open on more than one node of a cluster at the same time.

shared VM disk

A VM disk that belongs to a shared disk group in a cluster.

slave node

A node that is not designated as the master node of a cluster.

slice

The standard division of a logical disk device. The terms *partition* and *slice* are sometimes used synonymously.

snapshot

A point-in-time copy of a volume (volume snapshot) or a file system (file system snapshot).

spanning

A layout technique that permits a volume (and its file system or database) that is too large to fit on a single disk to be configured across multiple physical disks.

sparse plex

A plex that is not as long as the volume or that has holes (regions of the plex that do not have a backing subdisk).

Storage Area Network (SAN)
A networking paradigm that provides easily reconfigurable connectivity between any subset of computers, disk storage and interconnecting hardware such as switches, hubs and bridges.

stripe
A set of stripe units that occupy the same positions across a series of columns.

stripe size
The sum of the stripe unit sizes comprising a single stripe across all columns being striped.

stripe unit
Equally-sized areas that are allocated alternately on the subdisks (within columns) of each striped plex. In an array, this is a set of logically contiguous blocks that exist on each disk before allocations are made from the next disk in the array. A *stripe unit* may also be referred to as a *stripe element*.

stripe unit size
The size of each stripe unit. The default stripe unit size is 64KB. The stripe unit size is sometimes also referred to as the *stripe width*.

striping
A layout technique that spreads data across several physical disks using stripes. The data is allocated alternately to the stripes within the subdisks of each plex.

subdisk
A consecutive set of contiguous disk blocks that form a logical disk segment. Subdisks can be associated with plexes to form volumes.

swap area
A disk region used to hold copies of memory pages swapped out by the system pager process.

swap volume
A VxVM volume that is configured for use as a swap area.

transaction
A set of configuration changes that succeed or fail as a group, rather than individually. Transactions are used internally to maintain consistent configurations.

volboot file
A small file that is used to locate copies of the boot disk group configuration. The file may list disks that contain configuration copies in standard locations, and can also contain direct pointers to configuration copy locations. The `volboot` file is stored in a system-dependent location.

VM disk
A disk that is both under VxVM control and assigned to a disk group. VM disks are sometimes referred to as VxVM disks or simply disks.

volume

A virtual disk, representing an addressable range of disk blocks used by applications such as file systems or databases. A volume is a collection of from one to 32 plexes.

volume configuration device

The volume configuration device (`/dev/vx/config`) is the interface through which all configuration changes to the volume device driver are performed.

volume device driver

The driver that forms the virtual disk drive between the application and the physical device driver level. The volume device driver is accessed through a virtual disk device node whose character device nodes appear in `/dev/vx/rdsk`, and whose block device nodes appear in `/dev/vx/dsk`.

volume event log

The device interface (`/dev/vx/event`) through which volume driver events are reported to utilities.

vxconfigd

The VxVM configuration daemon, which is responsible for making changes to the VxVM configuration. This daemon must be running before VxVM operations can be performed.

Index

Symbols

/dev/vx/dmp directory 126
/dev/vx/rdmp directory 126
/etc/default/vxassist file 241, 390
/etc/default/vxdg defaults file 403
/etc/default/vxdg file 171
/etc/default/vxdisk file 81, 97
/etc/default/vxse file 448
/etc/fstab file 290
/etc/volboot file 212
/etc/vx/darecs file 212
/etc/vx/disk.info file 93
/etc/vx/dmppolicy.info file 148
/etc/vx/volboot file 186
/sbin/init.d/vxvm-recover file 395

A

A/A disk arrays 126
A/A-A disk arrays 126
A/P disk arrays 125
A/P-C disk arrays 126
A/PF disk arrays 126
A/PF-C disk arrays 126
A/PG disk arrays 126
A/PG-C disk arrays 126
access port 125
activation modes for shared disk groups 402
ACTIVE
 plex state 225
 volume state 265
active path attribute 146
ACTIVE state 311
Active/Active disk arrays 126
Active/Passive disk arrays 125
adaptive load-balancing 148
adaptiveminq load-balancing 148
adding disks 101
agile devices 77
alignment constraints 242
allocation
 site-based 432
allsites attribute 436
APM
 configuring 162
application volumes 32
array policy module (APM)
 configuring 162
array ports
 disabling for DMP 153
 displaying information about 142
 enabling for DMP 154
array support library (ASL) 83
ASL
 array support library 83
Asymmetric Active/Active disk arrays 126
ATTACHING state 311
attributes
 active 146
 autogrow 322, 325
 autogrowby 322
 cache 325
 cachesize 325
 comment 222, 234
 dcolen 69, 251, 357
 dgalign_checking 243
 displaying for rules 447
 drl 252, 281
 fastresync 251, 252, 293
 for specifying storage 244
 for Storage Expert 447
 hasdcolog 293
 highwatermark 322
 init 260
 len 222
 listing for rules 447
 loglen 253
 logtype 253
 maxautogrow 322
 maxdev 189
 mirdg 332
 mirvol 331
 name 221, 234
 ncachemirror 325

ndcomirror 251, 252, 357
ndcomirs 275, 321
newvol 330
nmirror 330
nomanual 146
nopreferred 146
plex 234
preferred priority 146
primary 147
putil 222, 234
secondary 147
sequential DRL 252
setting for paths 146
setting for rules 448
snapvol 327, 332
source 327, 332
standby 147
subdisk 221
syncing 319, 344
tutil 222, 234
auto disk type 81
autogrow
tuning 346
autogrow attribute 322, 325
autogrowby attribute 322
autotrespass mode 125

B

backups
created using snapshots 319
creating for volumes 303
creating using instant snapshots 319
creating using third-mirror snapshots 348
for multiple volumes 333, 352
implementing online 371
of disk group configuration 213
balanced path policy 148
base minor number 187
blocks on disks 29
boot disk group 167
bootdg 167
BROKEN state 311

C

c# 20, 78
c#t#d# 78
c#t#d# based naming scheme 78
cache attribute 325
cache objects
creating 322
enabling 323
listing snapshots in 345
caches
creating 322
deleting 347
finding out snapshots configured on 347
growing 347
listing snapshots in 345
removing 347
resizing 347
shrinking 347
stopping 347
used by space-optimized instant snapshots 309
cachesize attribute 325
Campus Cluster feature
administering 431
campus clusters
administering 431
serial split brain condition in 190
cascade instant snapshots 312
cascaded snapshot hierarchies
creating 337
categories
disks 83
CDS
alignment constraints 242
compatible disk groups 171
disk format 81
cds attribute 171
cdsdisk format 81
check_all policy 160
check_alternate policy 160
check_disabled policy 160
check_periodic policy 160
checkpoint interval 479
CLEAN
plex state 225
volume state 265
clone_disk flag 177
cloned disks 176
cluster functionality
enabling 518
shared disks 519
cluster protocol version
checking 428
upgrading 428

clusters
activating disk groups 403
activating shared disk groups 425
activation modes for shared disk groups 402
benefits 397
checking cluster protocol version 427
cluster-shareable disk groups 401
configuration 410
configuring exclusive open of volume by node 426
connectivity policies 404
converting shared disk groups to private 424
creating shared disk groups 422
designating shareable disk groups 401
detach policies 404
determining if disks are shared 421
forcibly adding disks to disk groups 423
forcibly importing disk groups 423
importing disk groups as shared 423
initialization 410
introduced 398
joining disk groups in 424
limitations of shared disk groups 409
listing shared disk groups 421
maximum number of nodes in 397
moving objects between disk groups 424
node abort 417
node shutdown 416
nodes 399
operation of vxconfigd in 414
operation of VxVM in 398
private disk groups 401
private networks 399
protection against simultaneous writes 402
reconfiguration of 410
resolving disk status in 404
setting disk connectivity policies in 425
setting failure policies in 426
shared disk groups 401
shared objects 402
splitting disk groups in 424
upgrading cluster protocol version 428
use of DMP in 132
use of MC/ServiceGuard with VxVM 410
vol_fmr_logsz tunable 480
volume reconfiguration 413
vxclustadm 411
vxdctl 420
vxrecover 428
vxstat 428
cluster-shareable disk groups in clusters 401
columns
changing number of 298
checking number in volume 454
in striping 38
mirroring in striped-mirror volumes 255
comment
plex attribute 234
subdisk attribute 222
concatenated volumes 35, 236
concatenated-mirror volumes
converting to mirrored-concatenated 300
creating 249
defined 45
recovery 237
concatenation 35
condition flags for plexes 228
configuration backup and restoration 213
configuration changes
monitoring using vxnotify 213
configuration copies for disk group 474
configuration database
checking number of copies 451
checking size of 451
copy size 166
in private region 80
listing disks with 178
metadata 177
reducing size of 195
configuring
shared disks 519
connectivity policies 404
setting for disk groups 425
controllers
checking for disabled 454
disabling for DMP 153
disabling in DMP 136
displaying information about 141
enabling for DMP 154
mirroring across 247, 255
number 20
specifying to vxassist 244
upgrading firmware 154
controllers, mirroring guidelines 514
converting disks 91
copymaps 69
copy-on-write
used by instant snapshots 307

crash dumps
 using VxVM volumes for 107
Cross-platform Data Sharing (CDS)
 alignment constraints 242
 disk format 81
CVM
 cluster functionality of VxVM 397

D

d# 20, 78
data change object
 DCO 69
data redundancy 42, 43, 46
data volume configuration 62
database replay logs and sequential DRL 61
databases
 resilvering 62
 resynchronizing 62
DCO
 adding to RAID-5 volumes 277
 adding version 0 DCOs to volumes 356
 adding version 20 DCOs to volumes 275
 calculating plex size for version 20 70
 considerations for disk layout 200
 creating volumes with version 0 DCOs
 attached 250
 creating volumes with version 20 DCOs
 attached 252
 data change object 69
 determining version of 277
 dissociating version 0 DCOs from volumes 358
 effect on disk group split and join 200
 log plexes 71
 log volume 69
 moving log plexes 277, 358
 reattaching version 0 DCOs to volumes 359
 removing version 0 DCOs from volumes 358
 specifying storage for version 0 plexes 357
 specifying storage for version 20 plexes 276
 used with DRL 60
 version 0 69
 version 20 69
 versioning 68
dcolen attribute 69, 251, 357
DCOSNP
 plex state 225
DDL 22
 Device Discovery Layer 85
decision support
 implementing 374
default disk group 167
defaultdg 167, 168
defaults
 for vxdisk 81, 97
description file with vxmake 259
detach policy
 global 405
 local 406
DETACHED
 plex kernel state 229
 volume kernel state 267
device discovery
 introduced 22
 partial 82
Device Discovery Layer 85
Device Discovery Layer (DDL) 22, 85
device files to access volumes 262, 518
device names 20, 77
 configuring persistent 93
device nodes
 controlling access for volume sets 366
 displaying access for volume sets 366
 enabling access for volume sets 365
 for volume sets 364
Device path not valid 117
devices
 adding foreign 89
 agile 77
 fabric 82
 metadevices 78
 pathname 78
dgalign_checking attribute 243
dgfailpolicy attribute 409
dirty bits in DRL 60
dirty flags set on volumes 59
dirty region logging. See DRL
dirty regions 482
disable failure policy 407
DISABLED
 plex kernel state 229
 volume kernel state 267
disabled paths 138
disk access records
 stored in /etc/vx/darecs 212
disk arrays
 A/A 126
 A/A-A 126
 A/P 125

A/P-C 126
A/PF 126
A/PF-C 126
A/PG 126
A/PG-C 126
Active/Active 126
Active/Passive 125
adding disks to DISKS category 87
adding vendor-supplied support package 84
Asymmetric Active/Active 126
defined 21
excluding support for 86
listing excluded 86
listing supported 85
listing supported disks in DISKS category 87
multipathed 22
re-including support for 86
removing disks from DISKS category 89
removing vendor-supplied support package 84
disk drives
variable geometry 515
disk duplexing 42, 255
disk groups
activating shared 425
activation in clusters 403
adding disks to 171
avoiding conflicting minor numbers on import 187
boot disk group 167
bootdg 167
checking for non-imported 452
checking initialized disks 452
checking number of configuration copies in 452
checking on disk config size 451
checking size of configuration database 451
checking version number 451
clearing locks on disks 186
cluster-shareable 401
compatible with CDS 171
configuration backup and restoration 213
configuring site consistency on 435
configuring site-based allocation on 434
converting to private 424
creating 170
creating shared 422
creating with old version number 212
default disk group 167
defaultdg 167
defaults file for shared 403
defined 28
deporting 173
designating as shareable 401
destroying 208
determining the default disk group 168
disabling 207
displaying boot disk group 168
displaying default disk group 168
displaying free space in 170
displaying information about 169
displaying version of 211
effect of size on private region 166
elimination of rootdg 165
failure policy 407
features supported by version 210
forcing import of 187
free space in 384
impact of number of configuration copies on performance 473
importing 174
importing as shared 423
importing forcibly 423
importing with cloned disks 176
joining 197, 206
joining in clusters 424
layout of DCO plexes 200
limitations of move, split, and join 199
listing objects affected by a move 200
listing shared 421
making site consistent 439
moving between systems 185
moving disks between 184, 203
moving licensed EMC disks between 203
moving objects between 196, 203
moving objects in clusters 424
names reserved by system 167
nodg 167
number of spare disks 454
private in clusters 401
recovering destroyed 208
recovery from failed reconfiguration 198
removing disks from 172
renaming 183
reorganizing 195
reserving minor numbers 187
restarting moved volumes 204, 205, 207
root 28
rootdg 28, 165

serial split brain condition 190
setting connectivity policies in clusters 425
setting default disk group 168
setting failure policies in clusters 426
setting number of configuration copies 474
shared in clusters 401
specifying to commands 167
splitting 196, 205
splitting in clusters 424
Storage Expert rules 451
upgrading version of 208, 211
version 208, 210
disk media names 28, 77
disk names 77
configuring persistent 93
disk sparing
Storage Expert rules 454
disk## 29, 78
disk##-## 29
diskdetpolicy attribute 409
diskgroup## 77
disks 83
adding 101
adding to disk groups 171
adding to disk groups forcibly 423
adding to DISKS category 87
array support library 83
auto-configured 81
categories 83
CDS format 81
changing default layout attributes 97
changing naming scheme 91
checking for failed 454
checking initialized disks not in disk group 452
checking number of configuration copies in disk group 452
checking proportion spare in disk group 454
clearing locks on 186
cloned 176
complete failure messages 384
configuring newly added 82
configuring persistent names 93
converting 91
default initialization values 97
determining failed 383
determining if shared 421
Device Discovery Layer 85
disabled path 138
discovery of by VxVM 83
disk access records file 212
disk arrays 21
displaying information 120, 121
displaying information about 120, 169
displaying spare 386
dynamic LUN expansion 108
EFI 81
enabled path 138
enabling 117
enclosures 23
excluding free space from hot-relocation use 388
failure handled by hot-relocation 380
formatting 96
handling duplicated identifiers 175
hot-relocation 379
HP format 81
initializing 90, 97
installing 96
invoking discovery of 84
layout of DCO plexes 200
listing tags on 177
listing those supported in JBODs 87
making available for hot-relocation 387
making free space available for hot-relocation use 389
marking as spare 387
media name 77
metadevices 78
mirroring volumes on 272
moving between disk groups 184, 203
moving disk groups between systems 185
moving volumes from 290
names 77
naming schemes 78
nopriv 80
number 20
obtaining performance statistics 470
OTHER_DISKS category 83
partial failure messages 383
postponing replacement 112
primary path 138
putting under control of VxVM 90
reinitializing 101
releasing from disk groups 208
removing 110, 112
removing from disk groups 172
removing from DISKS category 89
removing from pool of hot-relocation

spares 388
removing from VxVM control 112, 172
removing tags from 178
removing with subdisks 111, 112
renaming 119
replacing 112
replacing removed 115
reserving for special purposes 119
resolving status in clusters 404
scanning for 82
secondary path 138
setting connectivity policies in clusters 425
setting failure policies in clusters 426
setting tags on 177
simple 80
spare 384
specifying to vxassist 244
stripe unit size 515
tagging with site name 434
taking offline 118
UDID flag 175
unique identifier 175
unreserving 120
upgrading contoller firmware 154
VM 28
writing a new identifier to 176
DISKS category 83
adding disks 87
listing supported disks 87
removing disks 89
DMP
check_all restore policy 160
check_alternate restore policy 160
check_disabled restore policy 160
check_periodic restore policy 160
coexistence with native multipathing 130
configuring DMP path restoration policies 160
configuring I/O throttling 157
configuring response to I/O errors 156
disabling array ports 153
disabling controllers 153
disabling multipathing 133
disabling paths 153
displaying DMP database information 137
displaying DMP node for a path 139
displaying DMP node for an enclosure 140
displaying information about array ports 142
displaying information about controllers 141
displaying information about enclosures 142
displaying information about paths 137
displaying LUN group for a node 140
displaying paths controlled by DMP node 140
displaying paths for a controller 141
displaying paths for an array port 141
displaying recoveryoption values 159
displaying status of DMP error handling thread 162
displaying status of DMP path restoration thread 161
displaying TPD information 143
dynamic multipathing 125
enabling array ports 154
enabling controllers 154
enabling multipathing 134
enabling paths 154
enclosure-based naming 127
gathering I/O statistics 144
in a clustered environment 132
load balancing 129
logging levels 476
metanodes 126
migrating to or from native multipathing 130
nodes 126
path aging 476
path failover mechanism 128
path-switch tunable 476
renaming an enclosure 155
restore policy 160
scheduling I/O on secondary paths 151
setting the DMP restore polling interval 160
stopping the DMP restore daemon 161
vxdmpadm 139
dmp_cache_open tunable 475
dmp_daemon_count tunable 475
dmp_delayq_interval tunable 475
dmp_failed_io_threshold tunable 475
dmp_fast_recovery tunable 475
dmp_health_time tunable 476
dmp_log_level tunable 476
dmp_path_age tunable 476
dmp_pathswitch_blks_shift tunable 476
dmp_probe_idle_lun tunable 477
dmp_queue_depth tunable 477
dmp_restore_cycles tunable 477
dmp_restore_interval tunable 477
dmp_restore_policy tunable 478
dmp_retry_count tunable 478
dmp_retry_timeout tunable 478

dmp_scsi_timeout tunable 478
dmp_stat_interval tunable 479
DRL
 adding log subdisks 220
 adding logs to mirrored volumes 281
 checking existence of 450
 checking existence of mirror 450
 creating volumes with DRL enabled 252, 253
 determining if active 278
 determining if enabled 278
 dirty bits 60
 dirty region logging 60
 disabling 278
 enabling on volumes 275
 hot-relocation limitations 381
 log subdisks 61
 maximum number of dirty regions 482
 minimum number of sectors 482
 recovery map in version 20 DCO 69
 re-enabling 278
 removing logs from mirrored volumes 282
 removing support for 279
 sequential 61
 use of DCO with 60
drl attribute 252, 281
DRL guidelines 515
dumps
 using VxVM volumes for 107
duplexing 42, 255
dynamic LUN expansion 108

E

EFI disks 81
EMC arrays
 moving disks between disk groups 203
EMPTY
 plex state 226
 volume state 265
ENABLED
 plex kernel state 229
 volume kernel state 267
enabled paths, displaying 138
enclosure-based naming 23, 79, 91
 displayed by vxprint 94
 DMP 127
enclosures 23
 discovering disk access names in 94
 displaying information about 142
 issues with nopriv disks 94
 issues with simple disks 94
 mirroring across 255
 setting attributes of paths 146
error messages
 Association count is incorrect 419
 Association not resolved 419
 Cannot auto-import group 419
 Configuration records are inconsistent 419
 Disk for disk group not found 187
 Disk group has no valid configuration copies 186, 419
 Disk group version doesn't support feature 209
 Disk is in use by another host 186
 Disk is used by one or more subdisks 172
 Disk not moving, but subdisks on it are 200
 Duplicate record in configuration 419
 import failed 186
 No valid disk found containing disk group 186
 tmpsize too small to perform this relayout 55
 Volume has different organization in each mirror 286
 vxdg listmove failed 200
errord daemon 128
exclusive-write mode 402
exclusivewrite mode 402
explicit failover mode 126
Extensible Firmware Interface (EFI) disks 81

F

fabric devices 82
FAILFAST flag 128
failover 397, 398
failover mode 125
failure handled by hot-relocation 380
failure in RAID-5 handled by hot-relocation 380
failure policies 407
 setting for disk groups 426
FastResync
 checking if enabled on volumes 293
 disabling on volumes 293
 effect of growing volume on 73
 enabling on new volumes 251
 enabling on volumes 292
 limitations 74
 Non-Persistent 67
 Persistent 68, 70
 size of bitmap 479
 snapshot enhancements 305

use with snapshots 66
fastresync attribute 251, 252, 293
file systems
growing using vxresize 285
shrinking using vxresize 285
unmounting 290
fire drill
defined 432
testing 440
firmware
upgrading 154
FMR. See FastResync
foreign devices
adding 89
formatting disks 96
free space in disk groups 384
fullinst snapshot type 343
full-sized instant snapshots 307
creating 327
creating volumes for use as 323

G

GAB 410
global detach policy 405
Group Membership and Atomic Broadcast (GAB) 410
guidelines
DRL 515
mirroring 514
RAID-5 516

H

hasdcolog attribute 293
HFS file systems
resizing 285
highwatermark attribute 322
host failures 442
hostnames
checking 455
hot-relocation
complete failure messages 384
configuration summary 385
daemon 380
defined 75
detecting disk failure 380
detecting plex failure 380
detecting RAID-5 subdisk failure 380
excluding free space on disks from use by 388
limitations 381
making free space on disks available for use by 389
marking disks as spare 387
modifying behavior of 395
notifying users other than root 395
operation of 379
partial failure messages 383
preventing from running 395
reducing performance impact of recovery 395
removing disks from spare pool 388
Storage Expert rules 454
subdisk relocation 385
subdisk relocation messages 390
unrelocating subdisks 390
unrelocating subdisks using vxassist 392
unrelocating subdisks using vxdiskadm 391
unrelocating subdisks using vxunreloc 392
use of free space in disk groups 384
use of spare disks 384
use of spare disks and free space 384
using only spare disks for 390
vxrelocd 380
HP disk format 81
hpdisk format 81
HP-UX multipathing 77

I

I/O
gathering statistics for DMP 144
kernel threads 19
scheduling on secondary paths 151
throttling 128
use of statistics in performance tuning 469
using traces for performance tuning 472
I/O operations
maximum size of 480
I/O policy
displaying 147
example 151
specifying 147
I/O throttling 157
identifiers for tasks 267
idle LUNs 477
implicit failover mode 125
init attribute 260
initialization
default attributes 97
of disks 90, 97

initialization of disks 90
instant snapshots
backing up multiple volumes 333
cascaded 312
creating backups 319
creating for volume sets 334
creating full-sized 327
creating space-optimized 324
creating volumes for use as full-sized 323
displaying information about 342
dissociating 340
full-sized 307
improving performance of synchronization 345
reattaching 338
refreshing 337
removing 341
removing support for 279
restoring volumes using 340
space-optimized 309
splitting hierarchies 341
synchronizing 344
Intelligent Storage Provisioning (ISP) 32
intent logging 304
INVALID volume state 266
ioctl calls 480, 481
IOFAIL plex condition 228
IOFAIL plex state 226
ISP volumes 32

J

JBODs
adding disks to DISKS category 87
listing supported disks 87
removing disks from DISKS category 89

K

kernel states
for plexes 229
volumes 266

L

layered volumes
converting to non-layered 300
defined 51, 237
striped-mirror 43
layout attributes
changing for disks 97
layouts
changing default used by vxassist 243
left-symmetric 48
specifying default 243
types of volume 236
leave failure policy 407
left-symmetric layout 48
len subdisk attribute 222
LIF area 102
LIF LABEL record 102
link objects 311
linked break-off snapshots 311
creating 331
linked third-mirror snapshots
reattaching 339
load balancing 126
across nodes in a cluster 398
displaying policy for 147
specifying policy for 147
load-balancing
specifying policy for 147
local detach policy 406
lock clearing on disks 186
LOG plex state 226
log subdisks 515
associating with plexes 220
DRL 61
logdisk 251, 256, 257
logical units 125
loglen attribute 253
logs
adding DRL log 281
adding for RAID-5 283
adding sequential DRL logs 281
adding to volumes 274
checking for disabled 452
checking for multiple RAID-5 logs on same disk 449
RAID-5 50, 59
removing DRL log 282
removing for RAID-5 284
removing sequential DRL logs 282
resizing using vxvol 288
specifying number for RAID-5 256
usage with volumes 237
logtype attribute 253
LUN 125
LUN expansion 108

LUN group failover 126
LUN groups
displaying details of 140
LUNs
idle 477

M

maps
adding to volumes 274
usage with volumes 237
master node
defined 400
discovering 420
maxautogrow attribute 322
maxdev attribute 189
MC/ServiceGuard
use with VxVM in clusters 410
memory
granularity of allocation by VxVM 482
maximum size of pool for VxVM 483
minimum size of pool for VxVM 485
persistence of FastResync in 67
messages
complete disk failure 384
hot-relocation of subdisks 390
partial disk failure 383
metadata 177
metadevices 78
metanodes
DMP 126
migrating between DMP and native multipathing 130
minimum queue load balancing policy 150
minor numbers 187
mirbrk snapshot type 343
mirdg attribute 332
mirrored volumes
adding DRL logs 281
adding sequential DRL logs 281
changing read policies for 289
checking existence of mirrored DRL 450
checking existence of without DRL 450
configuring VxVM to create by default 272
creating 249
creating across controllers 247, 255
creating across enclosures 255
creating across targets 245
defined 236
dirty region logging 60
DRL 60
FastResync 60
FR 60
logging 60
performance 464
removing DRL logs 282
removing sequential DRL logs 282
snapshots 66
mirrored-concatenated volumes
converting to concatenated-mirror 300
creating 249
defined 43
mirrored-stripe volumes
benefits of 42
checking configuration 453
converting to striped-mirror 300
creating 254
defined 236
performance 465
mirroring
defined 42
guidelines 514
mirroring controllers 514
mirroring plus striping 43
mirrors
adding to volumes 271
boot disk 103
creating of VxVM root disk 104
creating snapshot 349
defined 33
removing from volumes 273
specifying number of 249
mirvol attribute 331
mirvol snapshot type 343
multipathing
disabling 133
displaying information about 137
enabling 134
native to HP-UX 77
Multi-Volume Support 361

N

names
changing for disk groups 183
defining for snapshot volumes 352
device 20, 77
disk 77
disk media 28, 77
plex 31

plex attribute 234
renaming disks 119
subdisk 29
subdisk attribute 221
VM disk 29
volume 31
naming scheme
changing for disks 91
changing for TPD enclosures 94
for disk devices 78
native multipathing 77, 130
ncachemirror attribute 325
ndcomirror attribute 251, 252, 357
ndcomirs attribute 275, 321
NEEDSYNC volume state 266
newvol attribute 330
nmirror attribute 329, 330
NODAREC plex condition 228
nodes
DMP 126
in clusters 399
maximum number in a cluster 397
node abort in clusters 417
requesting status of 420
shutdown in clusters 416
use of vxclustadm to control cluster functionality 411
NODEVICE plex condition 228
nodg 167
nomanual path attribute 146
non-autotrespass mode 126
non-layered volume conversion 300
Non-Persistent FastResync 67
nopreferred path attribute 146
nopriv disk type 80
nopriv disks
issues with enclosures 94

O

objects
physical 20
virtual 26
off-host processing 369, 397
OFFLINE plex state 226
online backups
implementing 371
online invalid status 120
online relayout
changing number of columns 298
changing region size 299
changing speed of 299
changing stripe unit size 298
combining with conversion 300
controlling progress of 299
defined 54
destination layouts 294
failure recovery 58
how it works 54
limitations 57
monitoring tasks for 299
pausing 299
performing 294
resuming 299
reversing direction of 300
specifying non-default 298
specifying plexes 298
specifying task tags for 298
temporary area 54
transformation characteristics 58
transformations and volume length 58
types of transformation 295
viewing status of 299
online status 120
ordered allocation 245, 251, 257
OTHER_DISKS category 83
overlapped seeks 514

P

parity in RAID-5 46
partial device discovery 82
partition size
displaying the value of 147
specifying 148
path aging 476
path failover in DMP 128
pathgroups
creating 134
paths
disabling for DMP 153
enabling for DMP 154
setting attributes of 146
performance
analyzing data 469
benefits of using VxVM 463
changing values of tunables 474
combining mirroring and striping 465
effect of read policies 466
examining ratio of reads to writes 471

hot spots identified by I/O traces 472
impact of number of disk group configuration copies 473
improving for instant snapshot synchronization 345
load balancing in DMP 129
mirrored volumes 464
monitoring 467
moving volumes to improve 470
obtaining statistics for disks 470
obtaining statistics for volumes 468
RAID-5 volumes 465
setting priorities 467
striped volumes 464
striping to improve 471
tracing volume operations 468
tuning large systems 473
tuning VxVM 472
using I/O statistics 469
persistent device name database 93
persistent device naming 93
Persistent FastResync 68, 69, 70
physical disks
adding to disk groups 171
clearing locks on 186
complete failure messages 384
determining failed 383
displaying information 120
displaying information about 120, 169
displaying spare 386
enabling 117
excluding free space from hot-relocation use 388
failure handled by hot-relocation 380
initializing 90
installing 96
making available for hot-relocation 387
making free space available for hot-relocation use 389
marking as spare 387
moving between disk groups 184, 203
moving disk groups between systems 185
moving volumes from 290
partial failure messages 383
postponing replacement 112
releasing from disk groups 208
removing 110, 112
removing from disk groups 172
removing from pool of hot-relocation spares 388
removing with subdisks 111, 112
replacing 112
replacing removed 115
reserving for special purposes 119
spare 384
taking offline 118
unreserving 120
physical objects 20
ping-pong effect 132
plex attribute 330
plex conditions
IOFAIL 228
NODAREC 228
NODEVICE 228
RECOVER 228
REMOVED 228
plex kernel states
DETACHED 229
DISABLED 229
ENABLED 229
plex states
ACTIVE 225
CLEAN 225
DCOSNP 225
EMPTY 226
IOFAIL 226
LOG 226
OFFLINE 226
SNAPATT 226
SNAPDIS 226
SNAPDONE 226
SNAPTMP 227
STALE 227
TEMP 227
TEMPRM 227
TEMPRMSD 228
plexes
adding to snapshots 353
associating log subdisks with 220
associating subdisks with 218
associating with volumes 229
attaching to volumes 229
changing attributes 234
changing read policies for 289
checking for detached 452
checking for disabled 452
comment attribute 234
complete failure messages 384

condition flags 228
converting to snapshot 351
copying 233
creating 223
creating striped 224
defined 30
detaching from volumes temporarily 231
disconnecting from volumes 230
displaying information about 224
dissociating from volumes 233
dissociating subdisks from 221
failure in hot-relocation 380
kernel states 229
limit on number per volume 467
maximum number of subdisks 481
maximum number per volume 31
mirrors 33
moving 232, 277, 358
name attribute 234
names 31
partial failure messages 383
putil attribute 234
putting online 231, 270
reattaching 231
recovering after correctable hardware failure 383
removing 233
removing from volumes 273
sparse 57, 219, 229, 232
specifying for online relayout 298
states 224
striped 38
taking offline 230, 270
tutil attribute 234
types 31
polling interval for DMP restore 160
prefer read policy 289
preferred plex
performance of read policy 466
read policy 289
preferred priority path attribute 146
primary path 125, 138
primary path attribute 147
priority load balancing 150
private disk groups
converting from shared 424
in clusters 401
private network
in clusters 399
private region
checking size of configuration database 451
configuration database 80
defined 80
effect of large disk groups on 166
public region 80
putil
plex attribute 234
subdisk attribute 222

R

RAID-0 38
RAID-0+1 42
RAID-1 42
RAID-1+0 43
RAID-5
adding logs 283
adding subdisks to plexes 219
checking existence of log 450
checking log is mirrored 450
checking size of log 450
guidelines 516
hot-relocation limitations 381
logs 50, 59
parity 46
removing logs 284
specifying number of logs 256
subdisk failure handled by hot-relocation 380
volumes 46
RAID-5 volumes
adding DCOs to 277
adding logs 283
changing number of columns 298
changing stripe unit size 298
checking existence of RAID-5 log 450
checking number of columns 453
checking RAID-5 log is mirrored 450
checking size of RAID-5 log 450
creating 256
defined 236
performance 465
removing logs 284
raw device nodes
controlling access for volume sets 366
displaying access for volume sets 366
enabling access for volume sets 365
for volume sets 364
read policies
changing 289

performance of 466
prefer 289
round 289
select 289
siteread 289, 433, 434, 436
split 289
read-only mode 402
readonly mode 402
RECOVER plex condition 228
recovery
checkpoint interval 479
I/O delay 479
preventing on restarting volumes 271
recovery accelerator 62
recovery time
Storage Expert rules 449
redo log configuration 63
redundancy
checking for volumes 452
of data on mirrors 236
of data on RAID-5 236
redundant-loop access 25
region 80
regionsize attribute 275, 321, 322
reinitialization of disks 101
relayout
changing number of columns 298
changing region size 299
changing speed of 299
changing stripe unit size 298
combining with conversion 300
controlling progress of 299
limitations 57
monitoring tasks for 299
online 54
pausing 299
performing online 294
resuming 299
reversing direction of 300
specifying non-default 298
specifying plexes 298
specifying task tags for 298
storage 54
transformation characteristics 58
types of transformation 295
viewing status of 299
relocation
automatic 379
complete failure messages 384
limitations 381
partial failure messages 383
Remote Mirror feature
administering 431
remote mirrors
administering 431
REMOVED plex condition 228
removing disks 112
removing physical disks 110
replacing disks 112
replay logs and sequential DRL 61
REPLAY volume state 266
resilvering
databases 62
restoration of disk group configuration 213
restore policy
check_all 160
check_alternate 160
check_disabled 160
check_periodic 160
restored daemon 128
restrictions
VxVM-bootable volumes 103
resyncfromoriginal snapback 317
resyncfromreplica snapback 317
resynchronization
checkpoint interval 479
I/O delay 479
of volumes 59
resynchronizing
databases 62
root disk
creating mirrors 104
root disk group 28, 165
root disks
creating LVM from VxVM 106
creating VxVM 104
removing LVM 105
root mirrors
checking 454
root volumes
booting 104
rootability 102
checking 454
rootdg 28
round read policy 289
round-robin
load balancing 150
performance of read policy 466

read policy 289
rules
attributes 458
checking attribute values 447
checking disk group configuration copies 451
checking disk group version number 451
checking for full disk group configuration database 451
checking for initialized disks 452
checking for mirrored DRL 450
checking for multiple RAID-5 logs on a disk 449
checking for non-imported disk groups 452
checking for non-mirrored RAID-5 log 450
checking for RAID-5 log 450
checking hardware 454
checking hot-relocation 454
checking mirrored volumes without a DRL 450
checking mirrored-stripe volumes 453
checking number of configuration copies in disk group 452
checking number of RAID-5 columns 453
checking number of spare disks 454
checking number of stripe columns 454
checking on disk config size 451
checking plex and volume states 452
checking RAID-5 log size 450
checking rootability 454
checking stripe unit size 453
checking system name 455
checking volume redundancy 452
definitions 456, 457
finding information about 447
for checking hardware 454
for checking rootability 454
for checking system name 455
for disk groups 451
for disk sparing 454
for recovery time 449
for striped mirror volumes 453
listing attributes 447
result types 448
running 447
setting values of attributes 448

S

scandisks
vxdisk subcommand 82
secondary path 125
secondary path attribute 147
secondary path display 138
select read policy 289
sequential DRL
defined 61
maximum number of dirty regions 482
sequential DRL attribute 252
serial split brain condition 432
correcting 194
in campus clusters 190
in disk groups 190
shared disk groups
activating 425
activation modes 402
converting to private 424
creating 422
importing 423
in clusters 401
limitations of 409
listing 421
shared disks, configuring 519
shared-read mode 402
sharedread mode 402
shared-write mode 402
sharedwrite mode 402
simple disk type 80
simple disks
issues with enclosures 94
single active path policy 150
Site Awareness license 434
site consistency
configuring 435
defined 432
site failure
simulating 440
site failures
host failures 442
loss of connectivity 442
recovery from 440, 443
scenarios and recovery procedures 441
storage failures 442
site reattachment daemon 440
site storage class 436
site-based allocation
configuring for disk groups 434
configuring for volumes 436
defined 432
site-based consistency
configuring on existing disk groups 439

siteconsistent attribute 435
siteread read policy 289, 433, 434, 436
sites
 reattaching 440
size units 216
slave nodes
 defined 400
SmartSync 62
 disabling on shared disk groups 482
 enabling on shared disk groups 482
snap objects 72
snap volume naming 317
snapabort 305
SNAPATT plex state 226
snapback
 defined 306
 merging snapshot volumes 352
 resyncfromoriginal 317
 resyncfromreplica 317, 353
snapclear
 creating independent volumes 354
SNAPDIS plex state 226
SNAPDONE plex state 226
snapmir snapshot type 343
snapshot hierarchies
 creating 337
 splitting 341
snapshot mirrors
 adding to volumes 336
 removing from volumes 336
snapshots
 adding mirrors to volumes 336
 adding plexes to 353
 and FastResync 66
 backing up multiple volumes 333, 352
 backing up volumes online using 319
 cascaded 312
 comparison of features 65
 converting plexes to 351
 creating a hierarchy of 337
 creating backups using third-mirror 348
 creating for volume sets 334
 creating full-sized instant 327
 creating independent volumes 354
 creating instant 319
 creating linked break-off 331
 creating snapshots of 313
 creating space-optimized instant 324
 creating third-mirror break-off 329
 creating volumes for use as full-sized instant 323
 defining names for 352
 displaying information about 355
 displaying information about instant 342
 dissociating instant 340
 emulation of third-mirror 310
 finding out those configured on a cache 347
 full-sized instant 65, 307
 hierarchy of 312
 improving performance of synchronization 345
 linked break-off 311
 listing for a cache 345
 merging with original volumes 352
 of volumes 63
 on multiple volumes 317
 reattaching instant 338
 reattaching linked third-mirror 339
 refreshing instant 337
 removing 350
 removing instant 341
 removing linked snapshots from volumes 337
 removing mirrors from volumes 336
 restoring from instant 340
 resynchronization on snapback 317
 resynchronizing volumes from 353
 space-optimized instant 65, 309
 synchronizing instant 344
 third-mirror 64
 use of copy-on-write mechanism 307
snapstart 305
SNAPTMP plex state 227
snapvol attribute 327, 332
snapwait 330, 332
source attribute 327, 332
spaceopt snapshot type 343
space-optimized instant snapshots 309
 creating 324
spanned volumes 35
spanning 35
spare disks
 checking proportion in disk group 454
 displaying 386
 marking disks as 387
 used for hot-relocation 384
sparse plexes 57, 219, 229, 232
split read policy 289
STALE plex state 227

standby path attribute 147
states
for plexes 224
of link objects 311
volume 265
statistics gathering 128
storage
ordered allocation of 245, 251, 257
storage attributes and volume layout 244
storage cache
used by space-optimized instant snapshots 309
Storage Expert
check keyword 447
checking default values of rule attributes 447
command-line syntax 446
diagnosing configuration issues 449
info keyword 447
introduced 445
list keyword 447
listing rule attributes 447
obtaining a description of a rule 447
requirements 446
rule attributes 458
rule definitions 456, 457
rule result types 448
rules 446
rules engine 446
run keyword 447
running a rule 447
setting values of rule attributes 448
vxse 445
storage failures 442
storage processor 125
storage relayout 54
stripe columns 38
stripe unit size recommendations 515
stripe units
changing size 298
checking size 453
defined 38
striped plexes
adding subdisks 219
defined 38
striped volumes
changing number of columns 298
changing stripe unit size 298
checking number of columns 454
checking stripe unit size 453
creating 253
defined 236
failure of 38
performance 464
specifying non-default number of columns 254
specifying non-default stripe unit size 254
Storage Expert rules 453
striped-mirror volumes
benefits of 43
converting to mirrored-stripe 300
creating 254
defined 237
mirroring columns 255
mirroring subdisks 255
performance 465
trigger point for mirroring 255
stripe-mirror-col-split-trigger-pt 255
striping 38
striping guidelines 515
striping plus mirroring 42
subdisk names 29
subdisks
associating log subdisks 220
associating with plexes 218
associating with RAID-5 plexes 219
associating with striped plexes 219
blocks 29
changing attributes 221
comment attribute 222
complete failure messages 384
copying contents of 217
creating 215
defined 29
determining failed 383
displaying information about 216
dissociating from plexes 221
dividing 217
DRL log 61
hot-relocation 75, 379, 385
hot-relocation messages 390
joining 218
len attribute 222
listing original disks after hot-relocation 394
maximum number per plex 481
mirroring in striped-mirror volumes 255
moving after hot-relocation 390
moving contents of 217
name attribute 221
partial failure messages 383

physical disk placement 513
putil attribute 222
RAID-5 failure of 380
RAID-5 plex, configuring 516
removing from VxVM 221
restrictions on moving 217
specifying different offsets for unrelocation 393
splitting 217
tutil attribute 222
unrelocating after hot-relocation 390
unrelocating to different disks 393
unrelocating using vxassist 392
unrelocating using vxdiskadm 391
unrelocating using vxunreloc 392
swap space
using VxVM volumes to increase 107
SYNC volume state 266
synchronization
controlling for instant snapshots 344
improving performance of 345
syncing attribute 319, 344
syncpause 344
syncresume 344
syncstart 344
syncstop 344
syncwait 344
system names
checking 454, 455

T

t# 20, 78
tags
for tasks 267
listing for disks 177
removing from disks 178
removing from volumes 288
renaming 288
setting on disks 177
setting on volumes 257, 288
specifying for online relayout tasks 298
specifying for tasks 267
target IDs
number 20
specifying to vxassist 244
target mirroring 245, 255
task monitor in VxVM 267
tasks
aborting 268
changing state of 268, 269
identifiers 267
listing 268
managing 268
modifying parameters of 269
monitoring 268
monitoring online relayout 299
pausing 269
resuming 269
specifying tags 267
specifying tags on online relayout operation 298
tags 267
TEMP plex state 227
temporary area used by online relayout 54
TEMPRM plex state 227
TEMPRMSD plex state 228
third-mirror
snapshots 64
third-mirror break-off snapshots
creating 329
third-party driver (TPD) 84
throttling 128
TPD
displaying path information 143
support for coexistence 84
tpdmode attribute 94
trigger point in striped-mirror volumes 255
tunables
changing values of 474
dmp_cache_open 475
dmp_daemon_count 475
dmp_delayq_interval 475
dmp_failed_io_threshold 475
dmp_fast_recovery 475
dmp_health_time 476
dmp_log_level 476
dmp_path_age 476
dmp_pathswitch_blks_shift 476
dmp_probe_idle_lun 477
dmp_queue_depth 477
dmp_restore_cycles 477
dmp_restore_interval 477
dmp_restore_policy 478
dmp_retry_count 478
dmp_retry_timeout 478
dmp_scsi_timeout 478
dmp_stat_interval 479
vol_checkpt_default 479

vol_default_iodelay 479
vol_fmr_logsz 68, 479
vol_max_vol 480
vol_maxio 480
vol_maxioctl 480
vol_maxparallelio 481
vol_maxspecialio 481
vol_subdisk_num 481
volcvm_smartsync 482
voldrl_max_drtregs 482
voldrl_max_seq_dirty 61, 482
voldrl_min_regionsz 482
voliomem_chunk_size 482
voliomem_maxpool_sz 483
voliot_errbuf_dflt 483
voliot_iobuf_default 483
voliot_iobuf_limit 483
voliot_iobuf_max 484
voliot_max_open 484
volpagemod_max_memsz 484
volraid_minpool_size 485
volraid_rsrtransmax 485
tutil
plex attribute 234
subdisk attribute 222

U

UDID flag 175
udid_mismatch flag 176
units of size 216
use_all_paths attribute 151

V

V-5-1-2536 286
V-5-1-2829 209
V-5-1-552 172
V-5-1-569 419
V-5-1-587 186
V-5-2-3091 200
V-5-2-369 172
V-5-2-4292 200
version 0
of DCOs 69
version 20
of DCOs 69
versioning
of DCOs 68
versions
checking for disk group 451
disk group 208
displaying for disk group 211
upgrading 208
virtual objects 26
VM disks
defined 28
determining if shared 421
displaying spare 386
excluding free space from hot-relocation use 388
initializing 90
making free space available for hot-relocation use 389
marking as spare 387
mirroring volumes on 272
moving volumes from 290
names 29
postponing replacement 112
removing from pool of hot-relocation spares 388
renaming 119
vol## 31
vol##-## 31
vol_checkpt_default tunable 479
vol_default_iodelay tunable 479
vol_fmr_logsz tunable 68, 479
vol_max_vol tunable 480
vol_maxio tunable 480
vol_maxioctl tunable 480
vol_maxparallelio tunable 481
vol_maxspecialio tunable 481
vol_subdisk_num tunable 481
volbrk snapshot type 343
volcvm_smartsync tunable 482
voldrl_max_drtregs tunable 482
voldrl_max_seq_dirty tunable 61, 482
voldrl_min_regionsz tunable 482
voliomem_chunk_size tunable 482
voliomem_maxpool_sz tunable 483
voliot_errbuf_dflt tunable 483
voliot_iobuf_default tunable 483
voliot_iobuf_limit tunable 483
voliot_iobuf_max tunable 484
voliot_max_open tunable 484
volpagemod_max_memsz tunable 484
volraid_minpool_size tunable 485
volraid_rsrtransmax tunable 485
volume kernel states

DETACHED 267
DISABLED 267
ENABLED 267
volume length, RAID-5 guidelines 516
volume resynchronization 59
volume sets
adding volumes to 362
administering 361
controlling access to raw device nodes 366
creating 362
creating instant snapshots of 334
displaying access to raw device nodes 366
enabling access to raw device nodes 365
listing details of 363
raw device nodes 364
removing volumes from 364
starting 363
stopping 363
volume states
ACTIVE 265
CLEAN 265
EMPTY 265
INVALID 266
NEEDSYNC 266
REPLAY 266
SYNC 266
volumes
accessing device files 262, 518
adding DRL logs 281
adding logs and maps to 274
adding mirrors 271
adding RAID-5 logs 283
adding sequential DRL logs 281
adding snapshot mirrors to 336
adding subdisks to plexes of 219
adding to volume sets 362
adding version 0 DCOs to 356
adding version 20 DCOs to 275
advanced approach to creating 238
assisted approach to creating 239
associating plexes with 229
attaching plexes to 229
backing up 303
backing up online using snapshots 319
block device files 262, 518
booting VxVM-rootable 104
changing layout online 294
changing number of columns 298
changing read policies for mirrored 289
changing stripe unit size 298
character device files 262, 518
checking for disabled 452
checking for stopped 452
checking if FastResync is enabled 293
checking redundancy of 452
combining mirroring and striping for performance 465
combining online relayout and conversion 300
concatenated 35, 236
concatenated-mirror 45, 237
configuring exclusive open by cluster node 426
configuring site consistency on 435
configuring site-based allocation on 436
converting between layered and non-layered 300
converting concatenated-mirror to mirrored-concatenated 300
converting mirrored-concatenated to concatenated-mirror 300
converting mirrored-stripe to striped-mirror 300
converting striped-mirror to mirrored-stripe 300
creating 238
creating concatenated-mirror 249
creating for use as full-sized instant snapshots 323
creating from snapshots 354
creating mirrored 249
creating mirrored-concatenated 249
creating mirrored-stripe 254
creating RAID-5 256
creating snapshots 350
creating striped 253
creating striped-mirror 254
creating using vxmake 258
creating using vxmake description file 259
creating with version 0 DCOs attached 250
creating with version 20 DCOs attached 252
defined 31
detaching plexes from temporarily 231
disabling FastResync 293
disconnecting plexes 230
displaying information 264
displaying information about snapshots 355
dissociating plexes from 233
dissociating version 0 DCOs from 358
DRL 515

effect of growing on FastResync maps 73
enabling FastResync on 292
enabling FastResync on new 251
excluding storage from use by vxassist 244
finding maximum size of 242
finding out maximum possible growth of 284
flagged as dirty 59
initializing contents to zero 261
initializing using vxassist 260
initializing using vxvol 261
kernel states 266
layered 43, 51, 237
limit on number of plexes 31
limitations 31
making immediately available for use 260
maximum number of 480
maximum number of data plexes 467
merging snapshots 352
mirrored 42, 236
mirrored-concatenated 43
mirrored-stripe 42, 236
mirroring across controllers 247, 255
mirroring across targets 245, 255
mirroring all 272
mirroring on disks 272
mirroring VxVM-rootable 103
moving from VM disks 290
moving to improve performance 470
names 31
naming snap 317
obtaining performance statistics 468
performance of mirrored 464
performance of RAID-5 465
performance of striped 464
performing online relayout 294
placing in maintenance mode 270
preparing for DRL and instant snapshot operations 275
preventing recovery on restarting 271
RAID-0 38
RAID-0+1 42
RAID-1 42
RAID-1+0 43
RAID-10 43
RAID-5 46, 236
raw device files 262, 518
reattaching plexes 231
reattaching version 0 DCOs to 359
reconfiguration in clusters 413
recovering after correctable hardware failure 383
removing 290
removing DRL logs 282
removing from /etc/fstab 290
removing linked snapshots from 337
removing mirrors from 273
removing plexes from 273
removing RAID-5 logs 284
removing sequential DRL logs 282
removing snapshot mirrors from 336
removing support for DRL and instant snapshots 279
removing version 0 DCOs from 358
resizing 284
resizing using vxassist 286
resizing using vxresize 285
resizing using vxvol 287
restarting moved 204, 205, 207
restoring from instant snapshots 340
restrictions on VxVM-bootable 103
resynchronizing from snapshots 353
snapshots 63
spanned 35
specifying default layout 243
specifying non-default number of columns 254
specifying non-default relayout 298
specifying non-default stripe unit size 254
specifying storage for version 0 DCO plexes 357
specifying storage for version 20 DCO plexes 276
specifying use of storage to vxassist 244
starting 271
starting using vxassist 260
starting using vxvol 261
states 265
stopping 270
stopping activity on 290
striped 38, 236
striped-mirror 43, 237
striping to improve performance 471
taking multiple snapshots 317
tracing operations 468
trigger point for mirroring in striped-mirror 255
types of layout 236
upgrading to use new features 279
using logs and maps with 237

zeroing out contents of 261
vxassist
adding a log subdisk 220
adding a RAID-5 log 283
adding DCOs to volumes 357
adding DRL logs 281
adding mirrors to volumes 230, 271
adding sequential DRL logs 282
advantages of using 239
changing number of columns 298
changing stripe unit size 298
command usage 240
configuring exclusive access to a volume 426
configuring site consistency on volumes 435
converting between layered and non-layered volumes 300
creating cache volumes 322
creating concatenated-mirror volumes 249
creating mirrored volumes 249
creating mirrored-concatenated volumes 249
creating mirrored-stripe volumes 254
creating RAID-5 volumes 256
creating snapshots 348
creating striped volumes 253
creating striped-mirror volumes 254
creating volumes 239
creating volumes for use as full-sized instant snapshots 324
creating volumes with DRL enabled 253
creating volumes with version 0 DCOs attached 251
creating volumes with version 20 DCOs attached 252
defaults file 241
defining layout on specified storage 244
discovering maximum volume size 242
displaying information about snapshots 355
dissociating snapshots from volumes 354
excluding storage from use 244
finding out how much volumes can grow 284
listing tags set on volumes 257, 288
merging snapshots with volumes 353
mirroring across controllers 247, 255
mirroring across enclosures 255
mirroring across targets 245, 247
moving DCO log plexes 277
moving DCO plexes 358
moving subdisks after hot-relocation 392
moving volumes 470
relaying out volumes online 294
removing DCOs from volumes 280
removing DRL logs 282
removing mirrors 273
removing plexes 273
removing RAID-5 logs 284
removing tags from volumes 288
removing version 0 DCOs from volumes 358
removing volumes 290
replacing tags set on volumes 288
reserving disks 119
resizing volumes 286
resynchronizing volumes from snapshots 353
setting default values 241
setting tags on volumes 257, 288, 289
snapabort 305
snapback 306
snapshot 306
snapstart 305
specifing number of mirrors 249
specifying number of RAID-5 logs 256
specifying ordered allocation of storage 245
specifying plexes for online relayout 298
specifying storage attributes 244
specifying storage for version 0 DCO plexes 357
specifying tags for online relayout tasks 298
taking snapshots of multiple volumes 352
unrelocating subdisks after hot-relocation 392
vxcache
listing snapshots in a cache 345
resizing caches 347
starting cache objects 323
stopping a cache 347
tuning cache autogrow 346
vxcached
tuning 346
vxclustadm 411
vxconfigd
managing with vxdctl 212
monitoring configuration changes 213
operation in clusters 414
vxcp_lvm_root
used to create VxVM root disk 104
used to create VxVM root disk mirrors 104
vxdarestore
used to handle simple/nopriv disk failures 95
vxdco
dissociating version 0 DCOs from volumes 358

reattaching version 0 DCOs to volumes 359
removing version 0 DCOs from volumes 358
vxdctl
checking cluster protocol version 427
managing vxconfigd 212
setting a site tag 434
setting default disk group 168
upgrading cluster protocol version 428
usage in clusters 420
vxdctl enable
configuring new disks 82
invoking device discovery 84
used to configure new disks 82
vxddladm
adding disks to DISKS category 87
adding foreign devices 89
changing naming scheme 92
listing excluded disk arrays 86
listing supported disk arrays 85
listing supported disks in DISKS category 87
removing disks from DISKS category 89
used to exclude support for disk arrays 86
used to re-include support for disk arrays 86
vxdestroy_lvmroot
used to remove LVM root disks 105
vxdg
changing activation mode on shared disk groups 425
clearing locks on disks 186
configuring site consistency for a disk group 435
configuring site-based allocation for a disk group 434
controllingl CDS compatibility of new disk groups 171
converting shared disk groups to private 424
correcting serial split brain condition 195
creating disk groups 171
creating disk groups with old version number 212
creating shared disk groups 422
deporting disk groups 174
destroying disk groups 208
disabling a disk group 207
displaying boot disk group 168
displaying default disk group 168
displaying disk group version 211
displaying free space in disk groups 170
displaying information about disk groups 169
forcing import of disk groups 187
importing a disk group containing cloned disks 177
importing cloned disks 178
importing disk groups 175
importing shared disk groups 423
joining disk groups 206
listing disks with configuration database copies 178
listing objects affected by move 200
listing shared disk groups 421
listing spare disks 386
moving disk groups between systems 185
moving disks between disk groups 184
moving objects between disk groups 203
obtaining copy size of configuration database 166
placing a configuration database on cloned disks 177
reattaching a site 440
recovering destroyed disk groups 208
removing disks from disk groups 172
renaming disk groups 183
setting base minor number 188
setting disk connectivity policy in a cluster 425
setting disk group policies 409
setting failure policy in a cluster 426
setting maximum number of devices 189
simulating site failure 440
splitting disk groups 205
upgrading disk group version 211
vxdisk
clearing locks on disks 186
defaults file 81, 97
determining if disks are shared 421
discovering disk access names 94
displaying information about disks 169
displaying multipathing information 137
listing disks 120
listing spare disks 386
listing tags on disks 177
notifying dynamic LUN expansion 108
placing a configuration database on a cloned disk 177
removing tags from disks 178
scanning disk devices 82
setting a site name 434
setting tags on disks 177
updating the disk identifier 176

vxdisk scandisks
rescanning devices 82
scanning devices 82
vxdiskadd
adding disks to disk groups 171
creating disk groups 171
placing disks under VxVM control 101
vxdiskadm
Add or initialize one or more disks 97, 171
adding disks 97
adding disks to disk groups 171
Change/display the default disk layout 97
changing disk-naming scheme 91
changing the disk-naming scheme 91
creating disk groups 171
deporting disk groups 173
Disable (offline) a disk device 118
Enable (online) a disk device 117
Enable access to (import) a disk group 174
Exclude a disk from hot-relocation use 389
excluding free space on disks from hot-relocation use 389
importing disk groups 174
initializing disks 97
List disk information 121
listing spare disks 386
Make a disk available for hot-relocation use 389
making free space on disks available for hot-relocation use 389
Mark a disk as a spare for a disk group 387
marking disks as spare 387
Mirror volumes on a disk 272
mirroring volumes 272
Move volumes from a disk 291
moving disk groups between systems 187
moving disks between disk groups 185
moving subdisks after hot-relocation 391
moving subdisks from disks 172
moving volumes from VM disks 291
Remove a disk 110, 172
Remove a disk for replacement 113
Remove access to (deport) a disk group 173
removing disks from pool of hot-relocation spares 388
Replace a failed or removed disk 115
Turn off the spare flag on a disk 388
Unrelocate subdisks back to a disk 391
unrelocating subdisks after hot-relocation 391
vxdiskunsetup
removing disks from VxVM control 112, 172
vxdmpadm
changing TPD naming scheme 94
configuring an APM 162
configuring I/O throttling 157
configuring response to I/O errors 156
disabling controllers in DMP 136
disabling I/O in DMP 153
discovering disk access names 94
displaying APM information 162
displaying DMP database information 137
displaying DMP node for a path 140
displaying DMP node for an enclosure 140
displaying I/O error recovery settings 159
displaying I/O policy 147
displaying I/O throttling settings 159
displaying information about controllers 141
displaying information about enclosures 142
displaying partition size 147
displaying paths controlled by DMP node 140
displaying status of DMP error handling thread 162
displaying status of DMP restoration thread 161
displaying TPD information 143
enabling I/O in DMP 154
gathering I/O statistics 144
listing information about array ports 142
removing an APM 163
renaming enclosures 155
setting I/O policy 150, 151
setting path attributes 146
setting restore polling interval 160
specifying DMP path restoration policy 160
stopping DMP restore daemon 161
vxedit
changing plex attributes 234
changing subdisk attributes 221, 222
configuring number of configuration copies for a disk group 474
excluding free space on disks from hot-relocation use 388
making free space on disks available for hot-relocation use 389
marking disks as spare 387
removing a cache 347
removing disks from pool of hot-relocation spares 388

removing instant snapshots 341
removing plexes 234
removing snapshots from a cache 347
removing subdisks from VxVM 221
removing volumes 290
renaming disks 119
reserving disks 119
VxFS file system resizing 285
vxiod I/O kernel threads 19
vxmake
associating plexes with volumes 229
associating subdisks with new plexes 218
creating cache objects 322
creating plexes 223, 271
creating striped plexes 224
creating subdisks 215
creating volumes 258
using description file with 259
vxmend
putting plexes online 270
re-enabling plexes 231
taking plexes offline 230, 270
vxmirror
configuring VxVM default behavior 272
mirroring volumes 272
vxnotify
monitoring configuration changes 213
vxplex
adding RAID-5 logs 283
attaching plexes to volumes 229, 271
converting plexes to snapshots 351
copying plexes 233
detaching plexes temporarily 231
dissociating and removing plexes 233
dissociating plexes from volumes 234
moving plexes 232
reattaching plexes 231
removing mirrors 273
removing plexes 273
removing RAID-5 logs 284
vxprint
checking if FastResync is enabled 293
determining if DRL is enabled 278
displaying DCO information 276, 357
displaying plex information 224
displaying snapshots configured on a cache 347
displaying subdisk information 216
displaying volume information 264
enclosure-based disk names 94
identifying RAID-5 log plexes 284
listing spare disks 386
used with enclosure-based disk names 94
verifying if volumes are prepared for instant snapshots 321
viewing base minor number 188
vxrecover
preventing recovery 271
recovering plexes 383
restarting moved volumes 204, 205, 207
restarting volumes 271
vxrelayout
resuming online relayout 299
reversing direction of online relayout 300
viewing status of online relayout 299
vxrelocd
hot-relocation daemon 380
modifying behavior of 395
notifying users other than root 395
operation of 381
preventing from running 395
reducing performance impact of recovery 395
vxres_lvmroot
used to create LVM root disks 106
vxresize
growing volumes and file systems 285
limitations 285
shrinking volumes and file systems 285
vxrootmir
used to create VxVM root disk mirrors 105
vxsd
adding log subdisks 220
adding subdisks to RAID-5 plexes 219
adding subdisks to striped plexes 219
associating subdisks with existing plexes 219
dissociating subdisks 221
filling in sparse plexes 219
joining subdisks 218
moving subdisk contents 217
removing subdisks from VxVM 221
splitting subdisks 217
vxse
Storage Expert 445
vxse_dc_failures
rule to check for hardware failures 454
vxse_dg1
rule to check for full disk group configuration database 451

vxse_dg2
rule to check disk group configuration copies 451
vxse_dg3
rule to check on disk config size 451
vxse_dg4
rule to check disk group version number 451
vxse_dg5
rule to check number of configuration copies in disk group 452
vxse_dg6
rule to check for non-imported disk groups 452
vxse_disk
rule to check for initialized disks 452
vxse_disklog
rule to check for multiple RAID-5 logs on a disk 449
vxse_drl1
rule to check for mirrored volumes without a DRL 450
vxse_drl2
rule to check for mirrored DRL 450
vxse_host
rule to check system name 455
vxse_mirstripe
rule to check mirrored-stripe volumes 453
vxse_raid5
rule to check number of RAID-5 columns 453
vxse_raid5log1
rule to check for RAID-5 log 450
vxse_raid5log2
rule to check RAID-5 log size 450
vxse_raid5log3
rule to check for non-mirrored RAID-5 log 450
vxse_redundancy
rule to check volume redundancy 452
vxse_rootmir
rule to check rootability 454
vxse_spares
rule to check number of spare disks 454
vxse_stripes1
rule to check stripe unit size 453
vxse_stripes2
rule to check number of stripe columns 454
vxse_volplex
rule to check plex and volume states 452
vxsited daemon 440
vxsnap
adding snapshot mirrors to volumes 336
administering instant snapshots 307
backing up multiple volumes 333
controlling instant snapshot synchronization 344
creating a cascaded snapshot hierarchy 337
creating full-sized instant snapshots 327, 332
creating linked break-off snapshot volumes 331
creating space-optimized instant snapshots 325
displaying information about instant snapshots 342
dissociating instant snapshots 340
preparing volumes for DRL and instant snapshots operations 275
preparing volumes for instant snapshots 321
reattaching instant snapshots 338
reattaching linked third-mirror snapshots 339
refreshing instant snapshots 337
removing a snapshot mirror from a volume 336, 337
removing support for DRL and instant snapshots 279
restore 308
restoring volumes 340
splitting snapshot hierarchies 341
vxsplitlines
diagnosing serial split brain condition 194
vxstat
determining which disks have failed 383
obtaining disk performance statistics 470
obtaining volume performance statistics 468
usage with clusters 428
zeroing counters 469
vxtask
aborting tasks 269
listing tasks 269
monitoring online relayout 299
monitoring tasks 269
pausing online relayout 299
resuming online relayout 299
resuming tasks 269
vxtrace
tracing volume operations 468
vxtune
setting volpagemod_max_memsz 485
vxunreloc
listing original disks of hot-relocated subdisks 394

moving subdisks after hot-relocation 392
restarting after errors 394
specifying different offsets for unrelocated subdisks 393
unrelocating subdisks after hot-relocation 392
unrelocating subdisks to different disks 393
VxVM
benefits to performance 463
cluster functionality (CVM) 397
configuration daemon 212
configuring disk devices 82
configuring to create mirrored volumes 272
dependency on operating system 19
disk discovery 83
granularity of memory allocation by 482
limitations of shared disk groups 409
maximum number of data plexes per volume 467
maximum number of subdisks per plex 481
maximum number of volumes 480
maximum size of memory pool 483
minimum size of memory pool 485
objects in 26
operation in clusters 398
performance tuning 472
removing disks from 172
removing disks from control of 112
shared objects in cluster 402
size units 216
task monitor 267
types of volume layout 236
upgrading 208
upgrading disk group version 211
use with MC/ServiceGuard in clusters 410
VXVM_DEFAULTDG environment variable 167
VxVM-rootable volumes
mirroring 103
vxvol
configuring exclusive access to a volume 426
configuring site consistency on volumes 436
disabling DRL 278
disabling FastResync 293
enabling FastResync 292
initializing volumes 261
putting volumes in maintenance mode 270
re-enabling DRL 278
resizing logs 288
resizing volumes 287
restarting moved volumes 204, 205, 207
setting read policy 289
starting volumes 261, 271
stopping volumes 270, 290
zeroing out volumes 261
vxvset
adding volumes to volume sets 362
controlling access to raw device nodes 366
creating volume sets 362
creating volume sets with raw device access 365
listing details of volume sets 363
removing volumes from volume sets 364
starting volume sets 363
stopping volume sets 363

W

warning messages
Specified region-size is larger than the limit on the system 320

Z

zero
setting volume contents to 261